Massively multi-robot simulation in stage

Richard Vaughan

Received: 5 December 2007 / Accepted: 9 July 2008 / Published online: 22 August 2008

Abstract Stage is a C++ software library that simulates multiple mobile robots. Stage version 2, as the simulation backend for the Player/Stage system, may be the most commonly used robot simulator in research and university teaching today. Development of Stage version 3 has focused on improving scalability, usability, and portability. This paper examines Stage's scalability.

We propose a simple benchmark for multi-robot simulator performance, and present results for Stage. Run time is shown to scale approximately linearly with population size up to 100,000 robots. For example, Stage simulates 1 simple robot at around 1,000 times faster than real time, and 1,000 simple robots at around real time. These results suggest that Stage may be useful for swarm robotics researchers who would otherwise use custom simulators, with their attendant disadvantages in terms of code reuse and transparency.

Keywords Simulation · Swarm · Multi-robot · Stage · Player/stage

1 Introduction

Stage is a C++ software library that simulates populations of mobile robots. Stage version 2, as the simulation backend for the Player/Stage system, may be the most commonly used robot simulator in research and university teaching today. The author is the originator and lead developer of Stage. The development of Stage version 3 has focused on improving scalability, usability, and portability. This paper examines Stage's scalability.

This paper is the first description of using Stage for massively multi-robot experiments, suitable for swarm robotics and other research where the behavior of large robot populations is of interest. In a previous paper we described how an earlier version of the Player/Stage system was useful for experimenting with a few tens of simulated robots, with controllers

Electronic supplementary material The online version of this article (http://dx.doi.org/10.1007/s11721-008-0014-4) contains supplementary material, which is available to authorized users.

R. Vaughan (✉)
Simon Fraser University, Burnaby, British Columbia, Canada
e-mail: vaughan@sfu.ca

transferring easily to real robots (Gerkey et al. 2003). Here we focus on using Stage standalone (without Player) as a platform for much larger robot experiments. To help researchers decide whether Stage will be useful for their project, we propose a simple benchmark for multi-robot simulator performance, and present results for Stage in a range of simulation scenarios and population sizes. This paper introduces the current development version of Stage, which will be released as Stage version 3.0.0. This version has been substantially rewritten since the last release and published description. While it provides similar functionality to version 2, it is considerably faster.

Run time is shown below to scale approximately linearly with population size up to at least 100,000 simple robots. For example, Stage simulates 1 simple robot at around 1,000 times faster than real time, and 1,000 simple robots at around real time. These results suggest that Stage may be useful for swarm robotics researchers who would otherwise be using in-house custom simulators, with their attendant disadvantages in terms of code reuse and transparency.

1.1 Technical and methodological features

Stage's most important technical features are (i) it cooperates with Player (Gerkey et al. 2001), currently the most popular robot hardware interface, which allows offline development of code designed for real robots; (ii) it is relatively easy to use, yet (iii) it is realistic *enough* for many purposes, striking a useful balance between fidelity and abstraction that is different from many alternative simulators; (iv) it provides models of many of the common robot sensors; (v) it runs on Linux and other Unix-like platforms including Mac OS X, which is convenient for most roboticists; and (vi) it supports multiple robots sharing a world.

Stage also has some very important non-technical features: (i) Free Software license, the GNU General Public License version 2;[1] (ii) active community of users and developers worldwide; and (iii) status as a well-known platform. The positive feedback effect of these last two features is very important, as it creates the potential to improve research practice by encouraging replication of experiments and code reuse.

The increased performance of Stage now makes it potentially interesting to researchers who may have previously rejected using a general-purpose simulator due to lack of scalability. Using Stage (or a derivation of it) instead of building a custom simulator can save researcher time and money. But perhaps more important is the methodological advantage of using a well-known, free and open platform, with open source code. Openness encourages transparency, replication and modification of experiments—a vital part of the scientific method that is uncommon in our field due to the difficulty and cost of reimplementation.

1.2 Scope and outline

This paper is not intended to be a definitive guide to Stage's internals, nor is it intended to be a user guide, or even a comprehensive review of the state of the art in robot simulation. The aim is to concisely present the key features of Stage relevant to the swarm robotics community, including the first documented performance data and the public introduction of the Stage API.

The paper proceeds as follows. In Sect. 2 we present evidence for the claims made in this introduction. In Sect. 3 we identify the current alternatives to Stage. Sections 4 and 5 describe the key implementation details of the simulation engine that are relevant to large-population performance. Section 6 describes a minimal and prototypical "swarm robotics"

[1] http://www.gnu.org/licenses/old-licenses/gpl-2.0.html.

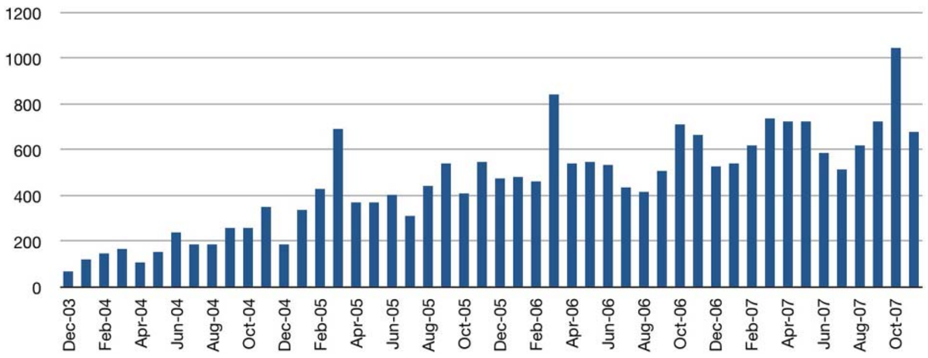

Fig. 1 Stage downloads from SourceForge by month, December 2003 to November 2007 (total 22,121)

robot controller and a set of simulation environments for benchmarking simulator performance. Section 7 provides benchmark results for Stage, followed by ideas for further improving performance and adding new features.

2 Evidence of stage impact

It is not practical to make a direct estimate of Stage installations or usage, as Stage's Free Software license allows free use and distribution. Most alternatives to Stage have distribution terms that make their use similarly uncountable, so we cannot know their relative popularity. Yet, Stage has a presence at the major research conferences and on university course web pages that suggests it is widely used and useful. For example, at least one paper using Stage for multi-robot simulations has appeared at ICRA, the main annual robotics meeting, every year since its debut in 2001 (Ye et al. 2001; Fredslund and Matarić 2002; Batalin and Sukhatme 2003; Shell and Matarić 2004; Batalin and Sukhatme 2004; Lin and Zheng 2005; Batalin and Sukhatme 2005; Chang et al. 2006; Busch et al. 2007). Only the first of these papers is by an author of Stage.

Player/Stage has been proposed as a "unifying paradigm to improve robotics education" (Anderson et al. 2007) and is used for classes at many universities (by professors who are not P/S authors) including Georgia Tech's CS3630, The University of Delaware's CISC 849, Washington University at St. Louis's CSE 550A, Brown University's CS 148, the University of Tennessee's CS 594, and the University of Southern California's CSCI 584.[2]

The "official" Stage distribution is maintained by the author at the Player Project website, hosted at SourceForge.[3] SourceForge tracks download statistics for each package, and the download history for Stage is shown in Fig. 1. Due to occasional failures of the statistics service, these numbers are (presumably slight) underestimates. Stage was downloaded from SourceForge at least 25,263 times between December 2001 and March 2008. Monthly downloads have grown steadily to a current rate of around 750 per month.

[2]Web pages for all these courses, describing assignments using Player/Stage, were available online at the time of writing (November 2007).

[3]http://playerstage.sourceforge.net.

3 Related work

An overview of the nature and goals of swarm robotics, and a survey of the field can be found in Dorigo and Şahin (2004), Şahin and Spears (2005). A survey and comparison of robot development platforms, including a limited discussion of simulators, is given in Kramer and Schultz (2007).

A good, detailed comparison of multi-robot simulators is overdue. A recent, flawed[4] short survey paper discussed some popular systems available in Craighead et al. (2007). We do not have space for a survey here, but the reader should be aware of the following influential systems.

- *TeamBots*:[5] This simple 2D Java-based multi-robot simulator was in popular use around 2000, due to its ease of use and free distribution terms, and the ability to run the same code in simulation and on real robots. One of the original Player/Stage goals was to reproduce the good features of TeamBots without the dependence on Java. TeamBots is still available, but development appears to have stopped in 2000 (Balch 1998).
- *Gazebo*:[6] A 3D simulator with dynamics from the Player Project, Gazebo is based on the Open Dynamics Engine.[7] Gazebo has become popular as a powerful, high-fidelity simulator that works with Player and has a GNU GPL license. However, it runs slowly with large populations (though no benchmarks are published) (Koenig and Howard 2004).
- *USARSim*:[8] Using the "Unreal" video game engine, USARSim is a GPL licensed 3D simulator that is similar in scope to Gazebo, Webots and Microsoft Robotics Studio. An important contribution of USARSim is its models of the NIST reference environments used for the RoboCup Urban Search and Rescue competition. Compatible with Player and MOAST (Scrapper et al. 2006), USARSim is under active development and has an active user community. USARSim aims for accurate dynamic models, and has been shown to support large worlds, though no data are available for large robot populations (Carpin et al. 2007).
- *Webots*:[9] This very high-quality commercial simulator focuses on accurate dynamical models of popular robots. It is fast (though no benchmarks are published), easy to use and has an attractive interface. Webots has a "Fast2DPlugin" extension that is optimized for simple, fast simulations, based on the Enki engine[10] and comes with detailed models of robots popular in swarm robotics such as the EPFL Alice, K-Team Khepera, Mondada's E-Puck robots, plus RoboCup soccer league models. Unfortunately, no scaling data are available, and Webots is not free to distribute or modify (Michel 2004).
- *Microsoft robotics studio*:[11] Released in early 2007, Microsoft Robotics Studio is functionally very similar to Player and Gazebo, though it works only on Microsoft's Windows

[4] Stage and Gazebo, two of the best known simulators, with different goals, scope, technology, scaling characteristics, authors, and user community, are conflated into one score, and surprisingly compared with Microsoft Flight Simulator (Craighead et al. 2007).

[5] http://www.cs.cmu.edu/~trb/TeamBots/.

[6] http://playerstage.sourceforge.net/index.php?src=gazebo.

[7] http://www.ode.org.

[8] http://usarsim.sourceforge.net.

[9] http://www.cyberbotics.com.

[10] http://home.gna.org/enki.

[11] http://msdn2.microsoft.com/en-us/robotics/default.aspx.

platform. Microsoft is promoting and financially supporting the use of Robotics Studio at some universities, notably Georgia Institute of Technology and Bryn Mawr College through the Institute for Personal Robots in Education.[12] Like Gazebo and Webots, Robotics Studio is based around a high-fidelity dynamics engine. Currently no examples of large population sizes seem to be available, but it is likely, provided Microsoft continues to support the project, that this system will be increasingly used in the coming years. Robotics studio is currently free to use, but not to modify or distribute, and the source code is not publicly available.

– *Swarmbot3d* (*S-bot simulator*):[13] The SWARM-BOTS project and s-bot robot system by Mondada, Dorigo, and colleagues is a highly successful and influential swarm robotics project. The project has its own **swarmbot3d** simulator, which necessarily includes dynamics as the s-bots interact with rough terrain and by pulling on each other. The simulator is possibly the most sophisticated considered here and is based on the Vortex™ commercial physics engine. Swarmbot3d is described in detail in (Mondada et al. 2004), including performance data for models at various levels of detail. It appears to run at speeds comparable to Stage, and appears to scale approximately linearly. Population sizes of 1 to 40 robots are reported. While Swarmbot3d is clearly a powerful tool, it is not publicly available.
– (*Swarmanoid simulator*):[14] The Swarmanoid project is the successor to the SWARM-BOTS project and examines heterogeneous swarms of robots. A distinguishing feature of ARGoS is its modular design, whereby robot controllers, sensors, actuators, physics engines, and visualizations are implemented as plug-ins. ARGoS provides multiple physics engines: 2D kinematics (like Stage), 2D dynamics, and 3D dynamics (like Gazebo, Robot Studio and Swarmbot3d). Uniquely, it is possible to run more than one physics engine in the same experiment. At the time of writing no benchmarks are published and ARGoS is not publicly available, though its authors plan to make it so [personal communication, May 2008].

Unlike most of the above simulators, Stage provides only first-order motion simulation, i.e., acceleration and momentum are not modeled. Dynamic physics simulation is required for many engineering modeling applications (and many video games). A short survey of physics engines is provided in Seugling and Rolin (2006), with additional references in Koenig and Howard (2004).

4 Stage internals for scaling

Stage's design philosophy has been described previously (Gerkey et al. 2003). The essential design feature that enables scaling is that the ray tracing computation required for collision detection and sensor modeling is approximately $O(1)$ per robot, i.e., it is *independent of the robot population size*. Ray tracing is by far the most frequent operation, performed possibly hundreds of times per robot per simulation timestep, and while other operations (such as hash table lookups) have worse theoretical growth characteristics, they occur relatively infrequently. Therefore, we can expect that Stage's overall run-time should grow almost linearly with population size.

[12] http://www.roboteducation.org/.

[13] http://www.swarm-bots.org/index.php?main=3&sub=33.

[14] http://www.swarmanoid.org/swarmanoid_simulation.php.

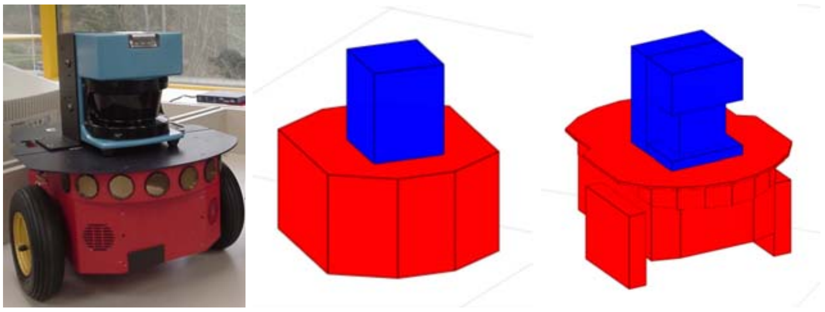

Fig. 2 Pioneer 3 DX robot (**a**) from MobileRobots Inc., and two virtual Stage robots, each composed of a StgModelPosition object (lower polyhedron) and a StgModelLaser object (upper polyhedron). The simple model (**b**) uses 2 blocks described by 16 numbers, and requires 12 rays for collision detection. The complex model (**c**) uses 11 blocks described by 98 numbers, and requires 74 rays for collision detection. These are our reference models for scaling experiments

Another performance bottleneck in the past was the graphical interface. This has been reimplemented using OpenGL. While the interface has become more sophisticated, now presenting 3-dimensional views, it has also gained a significant performance improvement due to highly optimized libraries, drivers, and hardware acceleration.

Here we outline how the ray tracing population independence is achieved, and introduce the OpenGL GUI.

4.1 Physical object model

Stage models physical bodies as tree assemblies of “blocks”. Blocks are specified as arbitrary polygons on the $[x, y]$-plane, extruded into the z-axis. The block data structure consists of a pointer to the model that owns the block, an array of two-dimensional points $[[x, y]_0, \ldots, [x, y]_n]$, the array length n, and a $[z^{\min}, z^{\max}]$ pair indicating the extent of the block in z. This implies that the blocks can be rotated only around the z-axis. Because of the missing degrees of freedom, and adopting terminology from computer graphics and video games, we describe Stage as a 2.5D (two-and-a-half dimensional) simulator.[15] One can think of blocks as Lego bricks, but with an arbitrary number of side faces.

Block trees can be constructed piecemeal by user code, specified in the worldfile with arrays of points, or loaded from bitmaps in most common file formats (JPEG, PNG, BMP, etc.). When interpreting bitmaps, Stage attempts to find a small number of blocks that occupy the same grid as the black pixels in the image.

Figure 2 shows two models supplied with Stage, both approximating the popular Pioneer 3DX robot. We compare the speed of Stage simulations using these models below.

4.2 Ray tracing

Collisions between blocks and range sensor data are computed using ray tracing. The population of 2.5D blocks is rendered into a 2-dimensional discrete occupancy grid by projecting their shapes onto the ground plane ($z = 0$). Grid cell size is configurable, with a default size

[15] http://en.wikipedia.org/wiki/2.5D.

of 0.02 m. Each grid cell contains a list of pointers to the blocks that have been rendered into that cell. When a block moves, it must be deleted from cells that it no longer occupies and rendered into newly-occupied cells. Non-moving blocks such as walls are rendered into the grid only once.

The ray tracing engine uses well-known techniques, so we summarize it only briefly. For speed and memory efficiency we use a two-level sparse nested grid data structure. Very fast bit-manipulation and integer arithmetic are used to look up grid cell locations from simulation world coordinates specified in meters. Ray tracing involves walking through the nested grid data structure using Cohen's integer line-drawing algorithm (Heckbert 1994). This was found empirically to be faster than the well-known floating-point alternative by Amanatides and Woo (1987). As the ray visits each non-empty cell, we inspect the z-extent of the blocks at that cell, and the properties of the models owning the block, to see if they interact with the ray.

We compared quadtree and kd-tree implementations with our nested grid, and found the grid to run considerably faster, probably due to better memory locality and thus better cache performance. This new ray tracing engine is the main source of Stage's recent performance improvements, and efforts to further improve performance should focus here.

4.3 User interface

New in Stage version 3 is a user interface using OpenGL[16] and the Fast Light Toolkit framework (FLTK),[17] chosen for speed, ease of use, and wide availability. The new graphics and user interface implementation is the second most significant performance improvement after ray tracing. The OpenGL interface provides a full 3D view of the world, and alpha blending and antialiasing provide scope for sophisticated effects for visualization of sensor data, motion history, etc.

Figures 4, 5, and 6 show screenshots from Stage demonstrating the new 3D view. OpenGL takes advantage of graphics processor (GPU) hardware, and we plan to take further advantage of the GPU and also to design more advanced sensor and robot state visualizations (see Future work section).

5 Using stage as a library

To date, Stage is most commonly used with Player, to form the Player/Stage system. Robot controllers are written as Player clients, and communicate with Player/Stage through network sockets. It is not well known that Stage version 2 is quite usable as a stand-alone C library, allowing users to embed a complete Stage simulation into their programs. This is done, for example, in MobileSim,[18] the simulator shipped by MobileRobots, Inc. to support their range of robots. MobileSim author Reed Hedges contributes bugfixes and features back to the Stage library. Until this paper, we have not publicized this way of using Stage. With the release of version 3, the Stage API will be "officially" documented and supported.

However, Player clients send and receive data asynchronously with Player through the host's networking subsystem. This requires significant interaction between the client and

[16] http://www.opengl.org.

[17] http://www.fltk.org.

[18] http://robots.mobilerobots.com/MobileSim.

```
#include <stage.hh>

int main( int argc, char* argv[] )
{
  StgWorld::Init( &argc, &argv );  // initialize libstage
  StgWorldGui world( 800, 600, "My Stage simulation");
  world.Load( argv[1] ); // load the world file

  // get a pointer to a mobile robot model
  // char array argv[2] must match a position model named in the worldfile
  StgModelPosition* robot =  (StgModelPosition*)world.GetModel( argv[2] );
  robot->Subscribe();

  world.Start();   // start the simulation clock running

  while( ! world.TestQuit() )
    if( world.RealTimeUpdate() )
    {
       // [ read sensors and decide actions here ]

       robot->Do( forward_speed, side_speed, turn_speed );
    }

  delete robot;
  exit( 0 );
}
```

Fig. 3 Runnable C++ source code for a minimal but complete Stage robot simulation. Error checking is omitted for brevity. The resulting compiled program could be run with the worldfile of Fig. 4 using: `./mystagesim minimal.world MySimpleRobot`. The resulting simulation would look like the screenshot in Fig. 4

server processes and the host operating system, with frequent context switches and likelihood of messages waiting in queues. This overhead may not be acceptable when attempting to scale robot populations. As the Player client–server link is asynchronous, it is not generally possible to run repeatable experiments. Interaction with the OS scheduler and ambient processes means that Player clients are subject to apparently stochastic time delays and will produce stochastic robot behavior. Some users (including the author) have found it difficult to maintain good enough synchronization between client and Player/Stage version 2 when the host machine is heavily loaded, for example, by a demanding Stage simulation (tens or hundreds of robots). In particular, the latency experienced by a client is not independent of robot population size, leading to poor robot behavior. Efforts are underway to solve this problem. For large scale populations, in cases where Player features (mainly portability to real robots, but also access to Player's library of well-known algorithms) we now recommend using Stage directly as a library. Stage is deterministic, so a simulation using only deterministic robot controllers will be perfectly repeatable. Sensor data and control commands are contained within a single running process, avoiding costly context switches and exploiting the CPU cache.

Stage version 3 is now a C++ library, which can be used to create a complete simulator by creating a single "StgWorld" object and accessing its methods. Rather than devote a lot of text to this subject, we provide a complete minimal instance of Stage program in Fig. 3, a matching world description file and a screenshot of the resulting world in Fig. 4. This should

```
# Simulation worlds are specified using a simple tree-structured
# configuration file, or by user programs making individual object
# creation and configuration function calls. This is an example of a
# simple configuration file

# size of the world in meters [x y z]
size [16 16 3]

# create and specify an environment of solid walls loaded from an image
model
(
  # size of the model in meters [x y z] - fill the whole world
  size3 [16 16 0.5]
  color "gray30"

  # draw a 1m grid over this model
  gui_grid 1

  # this model can not be moved around with the mouse
  gui_movemask 0

  # interpret this image file as an occupancy grid and
  # construct a set of blocks that fill this grid
  bitmap "bitmaps/cave.png"

  # model has a solid boundary wall surrounding the blocks
  boundary 1
)

# create and specify a mobile robot carrying a laser scanner
position
(
 name "MySimpleRobot"
 color "red"
 size3 [0.5 0.5 0.4]

 # pose of the model in the world [x y heading]
 pose [-2.551 -0.281 -39.206]

 laser() # default pose is at parent's origin facing forwards
)
```

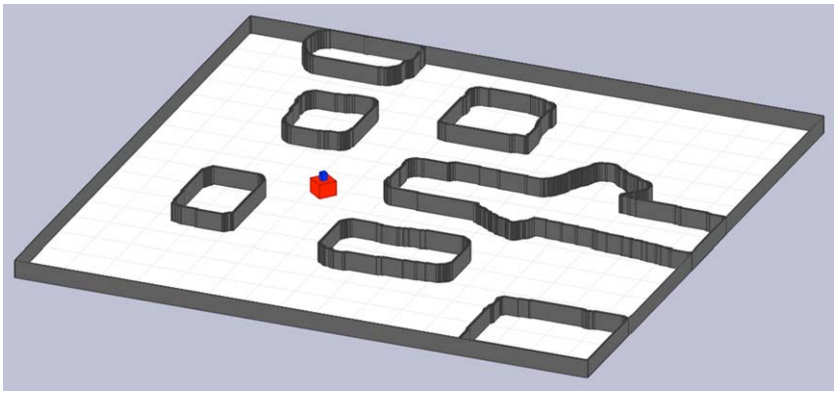

Fig. 4 Simple example of a Stage configuration file ("world file"), and a screenshot of the world it produces

be sufficient for a programmer to grasp the main ideas. Several examples are provided in the Stage distribution.

6 A benchmark for swarm simulation

We seek to obtain empirical run-time data for massively multi-robot simulation systems, in order to evaluate them and compare their performance. Direct comparison with other simulators is currently not possible as none have published benchmarks to date. However, even a Stage-only benchmark allows Stage users to evaluate the effect of their code additions and optimizations. We have been unable to find a suitable benchmark in the literature, so we propose the following simple procedure that should be possible to reproduce very closely on any comparable system.

Benchmark procedure A benchmark robot controller and world description are required, including a population of robot models. For simplicity, we run the identical robot controller concurrently on every robot, and measure the real time it takes to simulate a fixed amount of simulated time. To examine scalability, for each [controller, world description] pair, the run-time test should be performed for a single robot, then a series of exponentially increasing population sizes. For simulators or controllers with stochastic behavior, each experiment must be repeated several times to sample the distribution of run times.

6.1 Benchmark robot controller

It is unlikely that any single robot controller will satisfy every user, and any "real" controller that performs a particular useful swarm behavior from the literature could be argued to be too specialized. We prefer a controller that uses very little computation and memory compared to the simulator, to avoid controller costs masking the simulator performance. Yet, we want to see robot behavior which is somewhat representative of real swarm robotics research.

A canonical application for multi-robot systems, with approaches ranging from swarm intelligence to centralized planning, is wireless sensor network deployment and maintenance (see, e.g., Winfield 2000; Howard et al. 2006; Chang and Wang 2008; plus several ICRA articles cited above). This task is also the subject of past and present DARPA programs. Multi-robot deployment overlaps with other canonical tasks such as exploration and foraging, though it is not a good model of trail-following or cooperative manipulation.

A minimal version of the deployment problem is dispersal into the environment, which can be achieved using a mobile robot with n directional range sensors using the following trivial algorithm:

```
1. Gather array of range sensor vectors $\mathbf{v}_0, \dots, \mathbf{v}_{n-1}$
2. Compute vector sum of array, $\mathbf{w} = \sum_{m=0}^{n-1} \mathbf{v}_m$
3. IF(($current\_heading - direction\_of(\mathbf{w}) \approx 0$) AND (free space ahead))
   THEN (go forwards)
   ELSE (turn to reduce heading error)
4. GOTO 1
```

With suitable parameter choices depending on robot speed, number of sensors, maximum obstacle detection ranges, etc., this algorithm is highly effective at dispersing the robots until each robot has no object inside its sensor range. The algorithm is a simple approximation of Howard's approach (Howard et al. 2002), is trivial to implement, uses a single, commonly available class of sensor, and has the advantages of being stateless, thus requiring no per-robot storage between updates, and deterministic, avoiding the need for repeated trials.

Dispersal presents a worst-case scenario for grid-based simulations like Stage, as the amount of free space through which rays must propagate is maximized. Geometric simulations and those that track object bounding boxes may have an easier time, as the frequency of bounding box overlaps is minimized. If it is desirable to test the simulation performance for tightly clustered robots, the controller can trivially be inverted to produce local clustering.

We use this controller for the following benchmarking experiments, and hope that it may be useful for others or inspire (or provoke) the design of a superior method.

An evaluation approach that is often used in systems engineering is to have a family of benchmark controllers each capturing a different aspect of performance, and to present a vector of benchmark scores. In this context, our benchmark could be considered for a component of a future benchmark suite.

7 Stage benchmarks

We present benchmark results for Stage version 3 using multiple simulation scenarios. It is impractical to exhaustively explore the huge space of Stage configurations, but our choices span a range intended to be of interest to different types of user. The configurations described here should be straightforward to reproduce on most alternative simulators, and the files used to implement them will be included with the Stage version 3 distribution.

7.1 Ray tracing resolution

The spatial resolution of the underlying ray tracing engine was fixed for all tests at 0.02 m. This is Stage's default setting, and an informal survey of users suggests that few people ever change this parameter. This means that range sensors and collision detection is accurate to this value. By default, Stage contains no noise models for range data, so ranges are effectively quantized at 0.02 m. The justification for Stage's lack of sensor noise has been discussed elsewhere (Gerkey et al. 2003) and is based on the idea that Stage's ray tracing model imposes aliasing effects that play the role of noise, without the extra overhead. However, Stage version 3 comes with a an example plug-in sensor noise model that can be customized locally should the user desire particular noise effects.

7.2 Environments

We use two different environments created by the author, included for several years in the Stage distribution and used in papers by multiple researchers: the "Cave world" and the "Hospital world". The Cave environment was drawn by hand in 1998, and models a moderately constrained environment, either a crude artificial structure or a regular natural structure. The Hospital environment was created in 1999 by hand-editing a CAD drawing of the floorplan of the disused military Hospital at Fort Sam Houston, San Antonio, Texas, site of previous DARPA robot demonstrations. The bottom-left hand corner of the Hospital world is popularly used as a generic indoor environment, and is referred to as the "Hospital section".

Once loaded from their original bitmap files, the Cave consists of 489 blocks and is scaled to fill a 16 m by 16 m world. The Hospital consists of 5,744 blocks, and is approximately actual size at 140 m by 60 m. All blocks are rectangular polyhedra, with edges aligned with the global axes. The Cave and Hospital maps are shown in Figures 5 and 6, respectively.

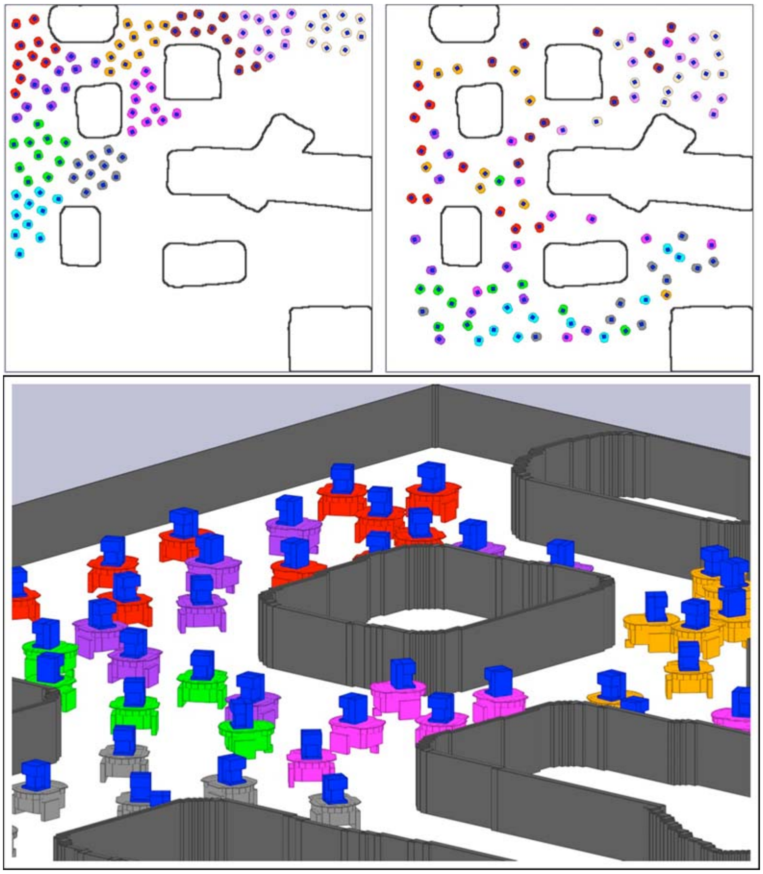

Fig. 5 The Cave world with 100 complex Pioneer models with lasers in the initial state (*top left*); after 600 simulated seconds (*top right*); and a 3D view of the center of the world at 60 seconds (*bottom*). A video of this simulation is available as Video 1–Cave world in the online supplementary material

7.3 Mobile robot models

We use the two mobile robot models shown in Fig. 2, similar but for the number and complexity of the blocks that make up their bodies. They are 0.44 m long by 0.38 m wide by 0.22 m high. Compared to the octagonal "simple" robot model, the "complex" robot model requires 6 times as many ray tracing operations to detect collisions (or lack of collisions) at each simulation update. It is also more expensive to draw in the GUI window, though the relative costs are hard to estimate without detailed knowledge of the specific OpenGL and rendering implementation. A third model, the "minibot" is used for larger scale experiments, and is similar in shape to the "simple" model, but is smaller, occupying a cube with sides of 0.2 m (shrunk to 0.1 m for the large-scale experiments reported in Table 3, to fit the robots into the world).

7.4 Sensor models

The "simple" and "complex" mobile robot models are identically equipped with generic Stage range finder models, 16 sensors distributed around the robot's boundary, each com-

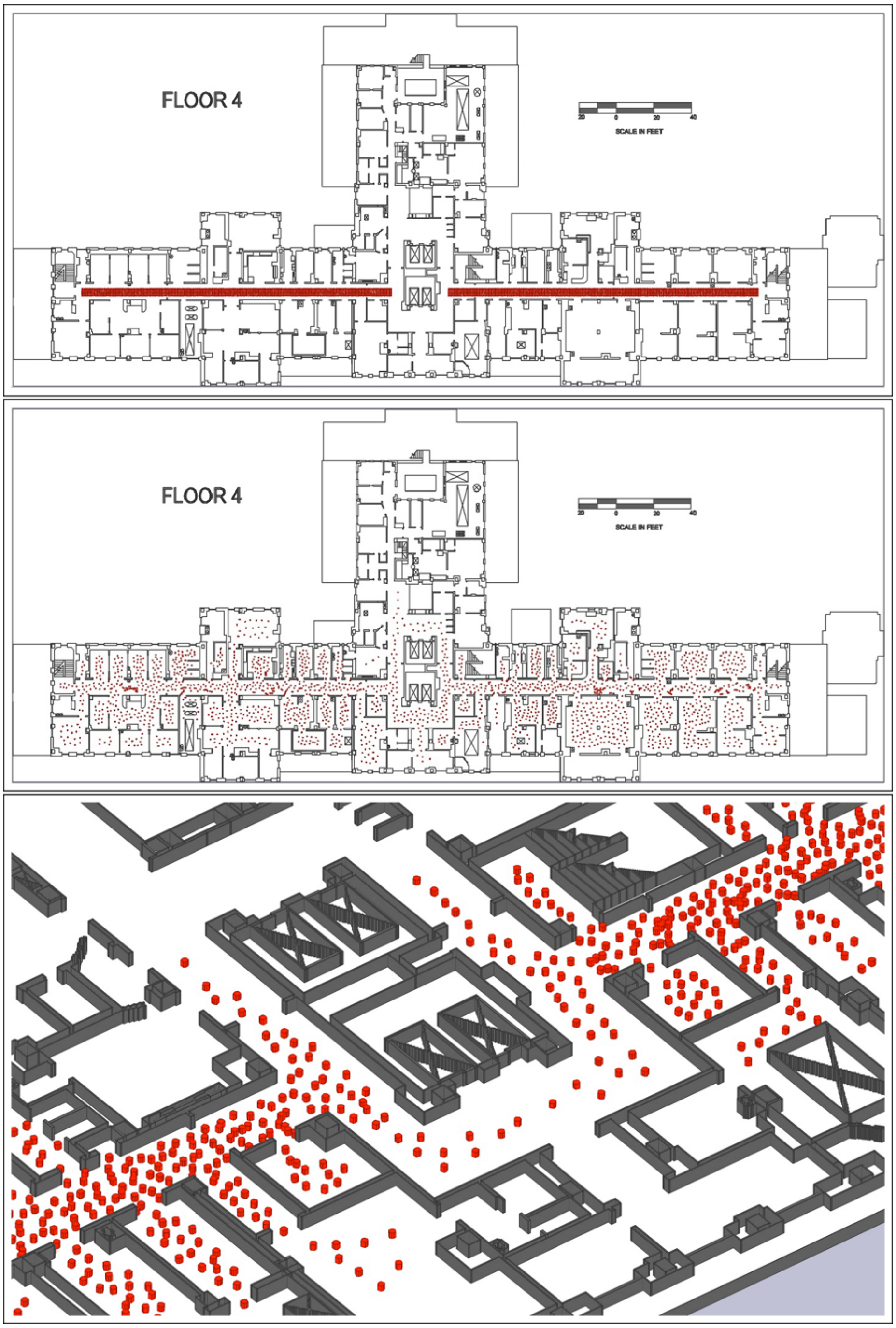

Fig. 6 The Hospital world with 2,000 minibots in its initial state (*top*) (robots are tightly packed and appear as a dark horizontal band); after 600 simulated seconds (*middle*); and a 3D view of the center of the world at 60 seconds (*bottom*). A video of this simulation is available as Video 2–Hospital world in the online supplementary material

Table 1 Cave world results: real-world run time for 600 seconds of simulation time, for the Cave world in 40 different configurations. Cave world is shown in Fig. 5. Simple and Complex models are shown in Fig. 2

Model type	Laser	Graphics	Run time for population size			
			1	10	50	100
Simple	–	–	0.55	4.98	24.78	46.23
″	–	yes	1.21	5.2	26.69	50.44
″	yes	–	4.06	42.70	229.48	449.44
″	yes	yes	4.83	45.71	243.28	476.34
Complex	–	–	1.23	12.03	59.15	116.69
″	–	yes	1.71	13.35	65.71	137.15
″	yes	–	5.21	53.61	278.47	550.11
″	yes	yes	6.05	58.44	315.83	659.66

puted by casting a single ray through free-space until a sonar-reflecting object is struck. Their maximum range matches the Pioneer robot's sonar sensors at 5 m.

The "minibot" has 12 range finders spaced evenly around its body, with a maximum range of 2 m. This models a long-range infrared range-finder or a short range sonar, such as is typically found on small robots designed for swarm experiments.

In addition, we test some scenarios with a model of a scanning laser rangefinder, parameterized to approximate the popular SICK LMS 200 device, except that it gathers 180 samples over its 180 degree field of view, instead of the 361 samples of the real device. The laser model has a maximum range of 8 m: the default on the real SICK. Due to its long range and large number of samples, the laser range model is much more computationally expensive than the sonar/infrared model.

7.5 Graphics

To examine the performance impact of the graphics rendering and user interface, the experiments were performed with and without the user interface. With the interface enabled, the window is repainted at the default interval of 100 milliseconds. We expect that Stage should run faster with the interface disabled.

7.6 Host computer

The host computer was an Apple MacBook Pro, with a 2.33 GHz Intel Core 2 Duo processor and 2 GB RAM, with a ATI Radeon X1600 graphics chipset with 256 MB VRAM, running Mac OS X 10.5.1. At the time of writing this is a high-end laptop, equivalent to a mid-range desktop PC.

7.7 Results

The benchmark robot controller was run in the Cave world for all 8 permutations of [model type, laser enabled/disabled, graphics enabled/disabled]. Each permutation was run for 600 seconds with population sizes of 1, 10, 50, and 100 robots. The real-world run times are recorded in Table 1. The success of the robot-dispersal algorithm can be seen in Fig. 5, which shows a global view of the Cave world with 100 complex robots at zero seconds and 600 seconds, and an intermediate 3D view of the center of the world at 60 seconds.

Table 2 Small-to-medium-scale Hospital world results: real-world run time for 600 seconds of simulation time, for the Hospital world in 10 different configurations. Hospital world and minibot model are shown in Fig. 6

Model type	Graphics	Run time for population size				
		1	10	100	1,000	2,000
Minibot (20 cm)	–	0.63	3.41	31.13	325.77	674.00
Minibot (20 cm)	yes	1.70	8.00	73.87	546.67	935.41

Table 3 Large-scale Hospital world results: real-world run time for 60 seconds of simulation time, for the Hospital world in 3 different configurations. Hospital world and minibot model are shown in Fig. 6

Model type	Graphics	Run time for population size		
		2,000	10,000	100,000
Minibot (10 cm)	yes	92.7	309.98	2,998.8

The benchmark robot controller was also run for 600 seconds in the Hospital world, with graphics enabled and disabled, for population sizes of 1, 10, 100, 1,000, and 2,000 robots. The real-world run times are recorded in Table 2. The success of the robot-dispersal controller can be seen again in Fig. 6, where 2,000 minibots are shown in the Hospital world at zero and 600 seconds, and in an intermediate 3D view at 60 seconds.

A final experiment investigates scaling to very large populations. We run 2,000, 10,000, and 100,000 minibots in the Hospital world. In order to fit the robots in the map, we shrink each robot's body to fit in a 10 cm cube. To keep the real-world run time reasonable, we run the simulation for only 60 simulated seconds. Otherwise this experiment is identical to its predecessor. The results are reported in Table 3. Again, we find that run time scales linearly with population size. However, with a population of 100,000 simple robots, Stage runs at around one-fiftieth of real time.

We see that, as expected, larger population sizes take longer to run. Use of the complex model and the laser scanner also increased run time, as we would expect from their ray tracing demands.

To compare the run time data between trials, we plot the ratio of simulated to real time (i.e., multiples of real time (MRT), or "speed-up factor") in Fig. 7. The population size is plotted against MRT on a log/log scale, and scenarios differing only in population size are connected with lines to form a scaling curve. We see that all the curves are approximately linear, indicating that Stage run time does increase linearly with population size as hoped. We also see that all but the six most demanding simulations ran faster than real time.

Figure 7 is rather cluttered, so we aim to clarify things by clustering the data. We observe that the differences in performance between experiments depends more on the sensors used than the robot body and the environment map. Grouping all experiments into one of the three sensor types used: (1) 16 sonar beams (range 5 m) + 180 laser beams (range 8 m); (2) 16 sonar beams (range 5 m); (3) 12 sonar beams (range 2 m), and plotting the mean real-world run time against population size, we obtain Fig. 8, which more clearly shows the roughly 10-fold performance cost of using the laser model.

To summarize the results, these data show that with a trivial robot controller, Stage can simulate small populations of complex robots much faster than real time. Populations of around 100 complex robots or 1,000 simple robots can run in real time or less. In addition,

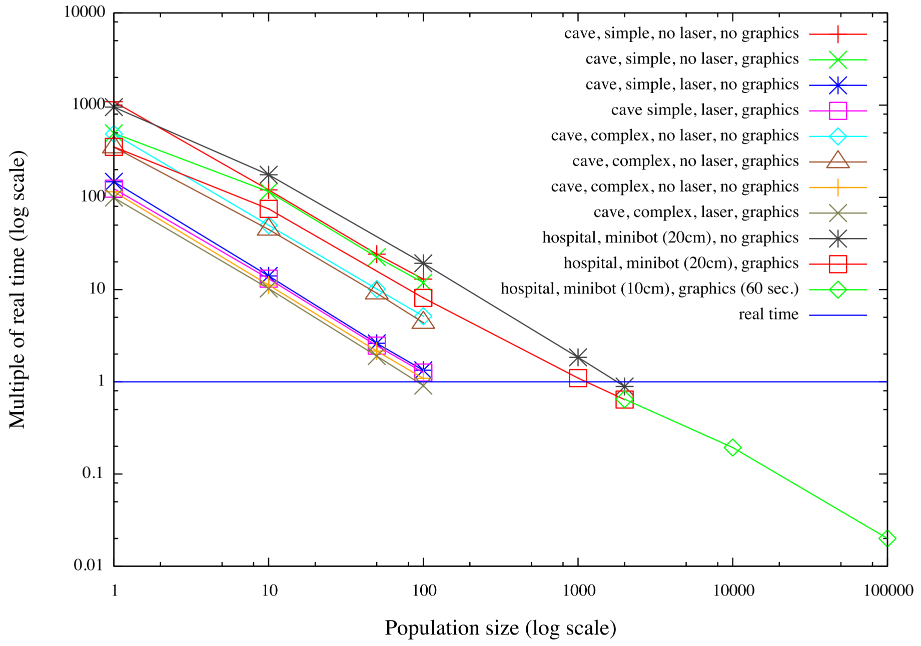

Fig. 7 Summary of benchmark results. The ratio of simulation time to real time is plotted against population size on a log/log scale, for each world configuration. Anything above the $x = 1$ line is running faster than real time. The run-time performance scales approximately linearly with population size in the range tested

we have shown that Stage scales to simulating up to 100,000 robots, though slower than real time. Long-duration experiments with such large populations, while possible, are still costly in terms of experimenter time.

8 Future work

8.1 Performance and scaling

Our 100,000 robot experiment ran at around one-fiftieth of real-time, which limits its usefulness for simulating long periods of time. Populations of 10,000 to 100,000 robots are in the regime of interesting natural swarms, so the swarm robotics community could benefit from a further speed increase. To achieve the required 10 to 100 times increase in performance to achieve real time performance with these populations, or to make simulating a million robots at all practical, without fundamentally changing our simulation model, we have three main areas of investigation, listed in reverse order of potential scaling benefit:

8.1.1 Optimize code efficiency

There is certainly scope for improvement in the performance of Stage's frequently run inner loops—mainly ray tracing and graphics rendering. In addition, to date little attention has been paid to per-robot memory size and layout. More careful memory use could improve cache hit rates, and allow larger populations before exceeding a host machine's physical memory.

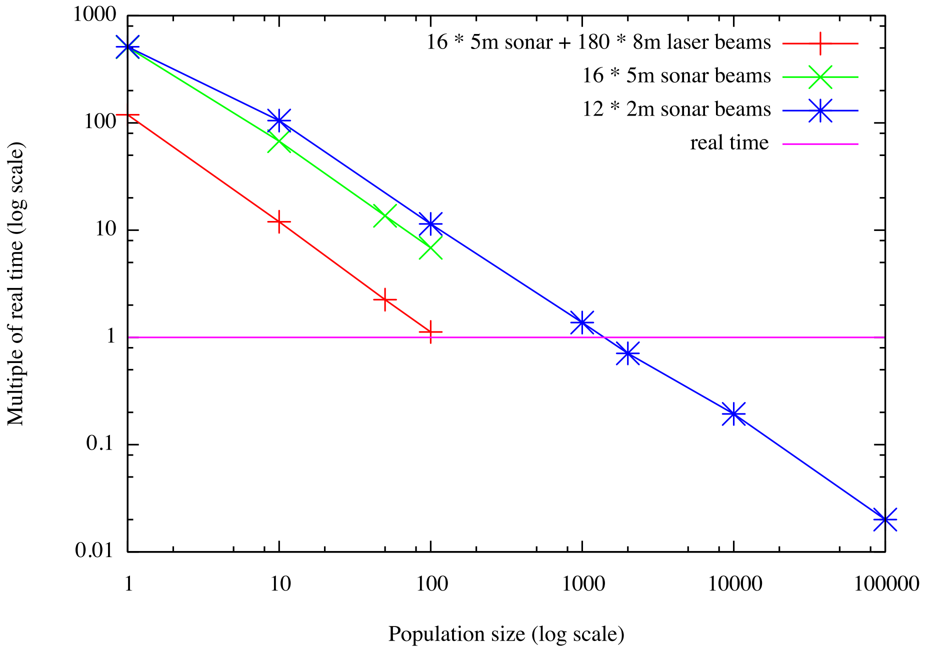

Fig. 8 Benchmark results clustered by sensor type and averaged. The mean ratio of simulation time to real time is plotted against population size on a log/log scale, for each of the following three types of sensor model: (1) $16*5$ m range sonar beams $+$ $180*8$ m range laser beams; (2) $16*5$ m range sonar beams; (3) $12*2$ m range sonar beams. The run-time performance scales approximately linearly with population size in the range tested

8.1.2 Concurrency—threads

Multi-core CPUs and multi-CPU machines have become common. In principle, Stage can benefit from parallel computation. Sections of the world that do not interact with each other can run concurrently without any synchronization overhead. Thus, we could, in principle, obtain a near N-times speed increase with N additional processor cores. Currently fast machines with 8 cores are available. The difficulty comes in maintaining suitable partitions, and finding linear time or better algorithms for partitioning physical simulations is an interesting area for research.

8.1.3 Concurrency—cluster computing

The greatest opportunity for massive scaling comes with cluster computing, in which many hosts can be pooled into a distributed system (e.g., Google's huge systems; see Barroso et al. 2003). Distributing Stage over a cluster is a similar problem to using threads, but with much greater between-partition communication costs, and the addition of reliability issues that can usually be ignored on single machines.

Both multi-thread and cluster implementations face additional overhead in maintaining a causal ordering of events to ensure that Stage simulations are repeatable in concurrent implementations. We plan to investigate a distributed implementation of Stage.

8.2 Other plans relevant to swarm robotics

8.2.1 Multi-robot visualization tools

We would like tools for analyzing, debugging, and presenting the state evolution of large-scale multi-robot systems. We wish to visualize the external and internal state of thousands of agents; to obtain insight into important events, causal relationships, patterns and problems. Exploiting the new OpenGL view of the world, we will gradually add visualizations to the main distribution. For example, Stage version 3 provides various views of the robot's movement history.

8.2.2 Standard API

Stage currently offers a custom API for creating simulations and writing robot controllers. It would be more useful if user's robot controllers were directly portable to Player, and other robot interfaces or simulators. The Player Project has begun work on creating a portable robot controller API, based on the Player Abstract Device Interface, to be implemented initially in Player and Stage. This is a long term community effort.

8.2.3 Portability

Stage version 3 is designed to be portable. It currently runs on most Unix-like systems, including Linux, OS X, Solaris, and the BSD family. It does not currently run out-of-the-box on Microsoft Windows, due to POSIX incompatibilities. We aim to have Stage build and run on Windows as standard in 2008. The main goal of this effort is to make Stage more accessible to students.

9 Critical limitations

While we have mentioned various limitations of Stage in the text above, it may be worthwhile to state the main issues all together, as follows.

Stage's key limitation is that it is a first-order geometric simulator: it ignores dynamics. It is a 2D simulator with some 3D extensions (commonly referred to as a 2.5D simulator). The ray tracing model is based on an occupancy tree, so the sensing and collision detection resolution is limited by the fixed size of the leaf cells. No bitmapped camera model is currently available, though this is planned for the near future. A simple odometry noise model is built in, and a demonstration sensor noise model is provided as an optional plugin, but otherwise Stage ignores sensor noise (relying on the aliasing effects of the low resolution ray tracing to provide a noise-like function). At the time of writing Stage does not run on Microsoft Windows.

10 Conclusion

We have argued that Stage may be useful to swarm robotics researchers because (i) it supports large numbers of robots at real time or better; and (ii) it is well known in the multi-robot systems community, and therefore has methodological advantages compared to a dedicated simulator.

To demonstrate the new performance of Stage version 3, we presented and used a simple benchmark for mobile robot simulators. Stage's run time was shown to scale approximately linearly with population size up to 100 complex robots or 100,000 simple robots. Populations of around 1,000 simple robots or 100 complex robots can be simulated in real time.

Stage is not useful for every mobile robot simulation project, but version 2 has proved itself useful to many. Stage version 3 adds a significant performance increase to this open, free and well-known platform.

Supporting files

All source code and configuration files required to reproduce these benchmarks are included in Stage source code distributions since Stage -3.0.0, in the directory `worlds/benchmark`. The configuration files are either human-readable text files or PNG format bitmaps. These files and this paper should be sufficient information to adapt the benchmark for another simulator.

Acknowledgements Thanks to Brian Gerkey, Toby Collett, Reed Hedges, Radu Rogan Rusu, Pooya Karimian, and the Player Project development community. Thanks to Andrew Howard for his work on earlier versions of Stage. Thanks to the editors and four anonymous reviewers for assistance in improving this paper. This work was supported by a Strategic Project Grant and a Discovery Grant from NSERC, and a research contract from DRDC in Canada.

References

Amanatides, J., & Woo, A. (1987). A fast voxel traversal algorithm for ray tracing. In *Proceedings of the conference of the European association for computer graphics (Eurographics '87)* (pp. 3–10). Amsterdam: Elsevier Science.

Anderson, M., Thaete, L., & Wiegand, N. (2007). Player/Stage: a unifying paradigm to improve robotics education delivery. In *Workshop on research in robots for education at robotics: science and systems conference*.

Balch, T. *Behavioral diversity in learning robot teams*. PhD thesis, College of Computing, Georgia Institute of Technology, 1998.

Barroso, L. A., Dean, J., & Holzle, U. (2003). Web search for a planet: the Google cluster architecture. *IEEE Micro*, *23*(2), 22–28.

Batalin, M. A., & Sukhatme, G. S. (2003). Efficient exploration without localization. In *Proceedings of the IEEE international conference on robotics and automation* (pp. 2714–2719). Los Alamitos: IEEE Computer Society Press.

Batalin, M. A., & Sukhatme, G. S. (2004). Using a sensor network for distributed multi-robot task allocation. In *Proceedings of the IEEE international conference on robotics and automation* (pp. 158–164). Los Alamitos: IEEE Computer Society Press.

Batalin, M., & Sukhatme, G. (2005). The analysis of an efficient algorithm for robot coverage and exploration based on sensor network deployment. In *Proceedings of the IEEE international conference on robotics and automation* (pp. 3478–3485). Los Alamitos: IEEE Computer Society Press.

Busch, M., Skubic, M., Keller, J., & Stone, K. (2007). A robot in a water maze: learning a spatial memory task. In *Proceedings of the IEEE international conference on robotics and automation* (pp. 1727–1732). Los Alamitos: IEEE Computer Society Press.

Carpin, S., Lewis, M., Wang, J., Balakirsky, S., & Scrapper, C. (2007). USARSim: a robot simulator for research and education. In *Proceedings of the IEEE international conference on robotics and automation* (pp. 1400–1405). Los Alamitos: IEEE Computer Society Press.

Chang, R. S., & Wang, S. H. (2008). Self-deployment by density control in sensor networks. *IEEE Transactions on Vehicular Technology*, *57*(3), 1745–1755.

Chang, H., Lee, C., Lu, Y., & Hu, Y. (2006). Simultaneous localization and mapping with environmental structure prediction. In *Proceedings of the IEEE international conference on robotics and automation* (pp. 4069–4074). Los Alamitos: IEEE Computer Society Press.

Craighead, J., Murphy, R., Burke, J., & Goldiez, B. (2007). A survey of commercial & open source unmanned vehicle simulators. In *Proceedings of the IEEE international conference on robotics and automation* (pp. 852–857). Los Alamitos: IEEE Computer Society Press.

Dorigo, M., & Şahin, E. (Eds.) Special issue: swarm robotics. *Autonomous Robots*, *17*(2–3), 2004.

Fredslund, J., & Matarić, M. J. (2002). Huey, Dewey, Louie, and GUI—commanding robot formations. In *Proceedings of the IEEE international conference on robotics and automation* (pp. 175–180). Los Alamitos: IEEE Computer Society Press.

Gerkey, B. P., Vaughan, R. T., Støy, K., Howard, A., Sukhatme, G., & Matarić, M. J. (2001). Most valuable player: a robot device server for distributed control. In *Proceedings of the IEEE/RSJ international conference on intelligent robotic systems* (pp. 1226–1231). Los Alamitos: IEEE Computer Society Press.

Gerkey, B., Vaughan, R. T., & Howard, A. (2003). The player/stage project: tools for multi-robot and distributed sensor systems. In *Proceedings of the 11th international conference on advanced robotics* (pp. 317–323). Los Alamitos: IEEE Computer Society Press.

Heckbert, P. (Ed). (1994). *Graphics gems IV*. Boston: Academic Press.

Howard, A., Matarić, M. J., & Sukhatme, G. S. (2002). Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. In *Proceedings of the international symposium on distributed autonomous robotic systems* (pp. 299–308). New York: Springer.

Howard, A., Parker, L. E., & Sukhatme, G. S. (2006). Experiments with large heterogeneous mobile robot team: exploration, mapping, deployment and detection. *International Journal of Robotics Research*, *25*(5), 431–447.

Koenig, N., & Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Proceedings of the IEEE/RSJ international conference on intelligent robots and systems* (pp. 2149–2154). Los Alamitos: IEEE Computer Society Press.

Kramer, J., & Schultz, M. (2007). Development environments for autonomous mobile robots: a survey. *Autonomous Robots*, *22*(2), 101–132.

Lin, L., & Zheng, Z. (2005). Combinatorial bids based multi-robot task allocation method. In *Proceedings of the IEEE international conference on robotics and automation* (pp. 1145–1150). Los Alamitos: IEEE Computer Society Press.

Michel, O. (2004). Webots: Professional mobile robot simulation. *International Journal of Advanced Robotic Systems*, *1*(1), 39–42.

Mondada, F., Pettinaro, G. C., Guignard, A., Kwee, I., Floreano, D., Deneubourg, J.-L., Nolfi, S., Gambardella, L., & Dorigo, M. (2004). SWARM-BOT: a new distributed robotic concept. *Autonomous Robots*, *17*(2-3), 193–221.

Şahin, E., & Spears, W. (2005). In *Lecture notes in computer science: Vol. 3342*. *Swarm robotics: SAB 2004 international workshop*, Revised Selected Papers Santa Monica, CA, USA, July 17, 2004. Heidelberg: Springer.

Scrapper, C., Balakirsky, S., & Messina, E. (2006). MOAST and USARSim–a combined framework for the development and testing of autonomous systems. In *Proceedings of the SPIE*. Bellingham: SPIE.

Shell, D., & Matarić, M. (2004). Directional audio beacon deployment: an assistive multi-robot application. In *Proceedings of the IEEE international conference on robotics and automation* (pp. 2588–2594). Los Alamitos: IEEE Computer Society Press.

Seugling, A., & Rolin, M. (2006). *Evaluation of physics engines and implementation of a physics module in a 3D authoring tool*. Department of Computer Science, Umeoa University, Sweden: Master's thesis.

Winfield, A.F.T. (2000). Distributed sensing and data collection via broken ad hoc wireless connected networks of mobile robots. In *Distributed autonomous robotic systems* (pp. 273–282). Heidelberg: Springer.

Ye, W., Vaughan, R. T., Sukhatme, G. S., Heidemann, J., Estrin, D., & Matarić, M. J. (2001). Evaluating control strategies for wireless-networked robots using an integrated robot and network simulation. In *Proceedings of the IEEE international conference on robotics and automation* (pp. 2941–2947). Los Alamitos: IEEE Computer Society Press.