Michael Gschwandtner, Roland Kwitt, Andreas Uhl and Wolfgang Pree, "BlenSor: Blender Sensor Simulation Toolbox", In G. Bebis, R. Boyle, B. Parvin, D. Koracin, R. Chung and R. Hammoud, editors, Advances in Visual Computing: 7th International Symposium, (ISVC 2011), Volume 6939/2011, pp. 199-208, Springer Verlag, 2011

BlenSor: Blender Sensor Simulation Toolbox

Michael Gschwandtner, Roland Kwitt, Andreas Uhl, Wolfgang Pree

Department of Computer Sciences, University of Salzburg, Austria
{mgschwan,rkwitt,uhl}@cosy.sbg.ac.at, wolfgang.pree@cs.uni-salzburg.at

Abstract. This paper introduces a novel software package for the simulation of various types of range scanners. The goal is to provide researchers in the fields of obstacle detection, range data segmentation, obstacle tracking or surface reconstruction with a versatile and powerful software package that is easy to use and allows to focus on algorithmic improvements rather than on building the software framework around it. The simulation environment and the actual simulations can be efficiently distributed with a single compact file. Our proposed approach facilitates easy regeneration of published results, hereby highlighting the value of reproducible research.

1 Introduction

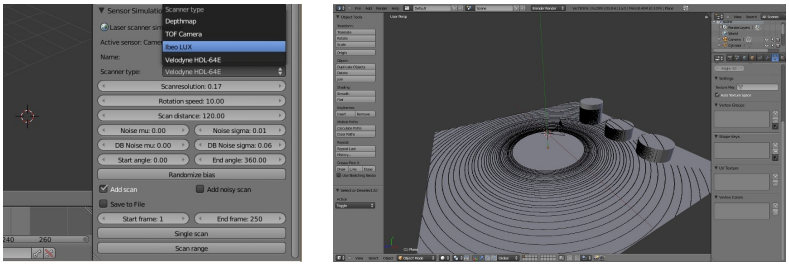
Light Detection and Ranging (LIDAR) devices are the key sensor technology in today's autonomous systems. Their output is used for obstacle detection, tracking, surface reconstruction or object segmentation, just to mention a few. Many algorithms exist which process and analyze the output of such devices. However, most of those algorithms are tested on recorded (usually not publicly available) sensor data and algorithmic evaluations rely on visual inspection of the results, mainly due to the lack of an available *ground truth*. Nevertheless, ground truth data is the key element to produce comparative results and facilitate a thorough quantitative analysis of the algorithms. Some authors tackle that problem by implementing their own sensor simulations, but most *home-brewed* approaches follow unrealistic simplifications, just using subdivision methods to generate point clouds for instance.

The software we propose in this article represents an approach to tackle that shortcoming: we provide a unified simulation and modeling environment which is capable of simulating several different types of sensors, carefully considering their special (physical) properties. This is achieved by integrating the simulation tool directly into Blender[1], a 3-D content creation suite. With this combination it is possible to model the test scenarios with arbitrary level of detail and immediately simulate the sensor output directly within the modeling environment. The *BlenSor*[2] toolkit is completely integrated within Blender (see Fig. 1a) and does not require any custom scripts or tedious editing of configuration files to adjust

[1] http://www.blender.org
[2] http://www.blensor.org

the sensors. Yet, it is possible to access the underlying scanning functionality from custom code in case researchers want to modify the core functionality.

The strong focus on offline data creation for algorithm development and testing allows BlenSor to focus on usability and features. BlenSor does not require to satisfy any external dependencies to enable compatibility with robotics frameworks for instance. The output is either i) written to a file (in a format explained in Section 3.6) or ii) added as a mesh within the sensor simulation. This facilitates direct interaction with the simulated (i.e *scanned*) data. Even though realtime capabilities have been left out on purpose, the simulation can be used together with Blender's physic engine, thus enabling to simulate complex scenarios with physical interaction of objects.



(a) Parameters **(b)** Exemplary scan simulation

Fig. 1: The sensor simulation interface is a part of the Blender GUI. It can be used just like any other feature of Blender: (a) every sensor has different parameters which can easily be modified and are stored in a `.blend` file; (b) example of a simple scan simulation. Single scans can be directly viewed and manipulated (and even analyzed) within Blender.

2 Previous Work

In [1], Dolson et al. generate range data for depth map upsampling by means of a custom OpenGL simulation. In [4], Meissner et al. simulate a four-layer laser range scanner using the ray-casting mechanism of the Blender game engine. Although, this is a fast and straightforward way of simulating a laser range scanner, it comes with the disadvantage of having to cope with restricted functionality of the game engine (e.g. limited set of materials, scalability issues, restrictions induced by graphics hardware, etc.). Bedkowski et al. [3] implement a custom simulation environment which provides an approximation of a laser scan performed by a LMS SICK 200. Their simulation however does not consider laser noise and is only a simulator which requires external modeling tools to create the scene that in turn is simulated. To the best of our knowledge, the most advanced simulation system is proposed by Echeverria et al. [2]. The authors provide an approach for realtime robotics simulation (named MORSE) using Blender as the underlying simulation environment. It supports several robotics frameworks and is meant for simulating the robots and studying their interaction with the

environment. The sensors, particularly the LIDAR types, are just a means to an end for simulation rather than the core component itself. In addition to that, simulation of the sensors is relatively limited in terms of physical correctness, i.e. no noise or reflections, and no Time-of-Flight camera is available as well.

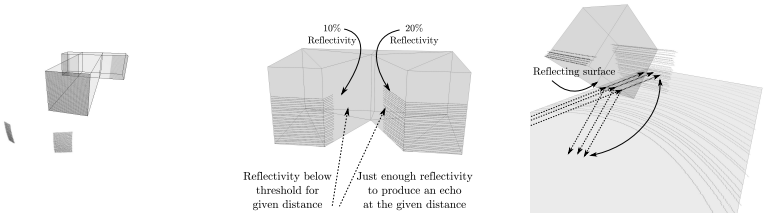
3 Sensor Simulation

Compared to robot simulation software ([2,7]), BlenSor focuses on simulation of the sensors itself rather than the interaction of sensor equipped robots with the environment. In fact, we are able to care a lot more about specific sensor properties, since there are no realtime constraints. Such properties are for example a realistic noise model, physical effects like reflection, refraction and reflectivity and sophisticated casting of rays that do not just describe a circle around the scanning center. The simulation accuracy can be increased with simple changes to the sensor code if features that are not yet available are required. The implementation details of the various sensor types in the following sections describe the simulation state at the time of writing. Due to the strong focus on offline simulation, we are able to simulate scenarios with a higher degree of detail than what is currently possible with existing robot simulators (e.g. MORSE ([2]).

3.1 Scanning Principle

All sensors simulated by BlenSor basically rely on the fact that the speed of light is finite and that light is at least partially reflected from most surfaces. To be more specific, the measured reflection is affected by i) the traveling distance of the emitted light, ii) the amount of light arriving at the sensor and iii) the concrete measurement time. In general, one or more rays of light are emitted from a range measurement device in the form of a light pulse. The rays travel along straight lines to a potential object. Once the rays hit an object, a fraction of the light gets reflected back to the sensor, some part gets reflected in different directions, and another part may pass through the object (in the case of transparent materials) in a possibly different direction.

This is in fact closely related to ray-tracing techniques in computer graphics. Thus the modification of a ray-tracing program to match the sensor characteristics seems just natural. Although Blender provides an interface to cast rays from within the Python programming language, the functionality is limited and runtime performance inevitably suffers due to the computational demand to simulate a huge number of laser rays. BlenSor tackles this problem by patching the Blender codebase to provide a way to cast several rays *simultaneously*. It also allows Pyhton code to access material properties of the faces that are hit by the rays. For increased efficiency, reflection is handled directly within Blender. By using this interface, the sensors developed using the Python interface, can set up an array of ray directions and hand the actual ray-casting over to the patched Blender core. Then, a raytree is built by Blender to allow efficient ray-casting. This modification processes all rays (and calculates reflections if needed) and returns the distances of the hits as well as the `objectID` for each ray. Eventually, the sensor code calculates sensor dependent noise and other physical features. This is described in the following sections.



(a) Backfolding **(b)** Low Reflectivity **(c)** Fully refl. surface

Fig. 2: Simulated features of different sensor types: (a) *Backfolding* effect of Time-of-Flight cameras; (b) Objects with low reflectivity (here: object in 50 meter distance); (c) Totally reflecting surfaces which cause points to appear farther away.

3.2 Rotating LIDAR

A rotating LIDAR has a sensor/emitter unit rotating around the center of gravity and thus creates a 360° scan of the environment. As a representative of this class of sensor type, BlenSor implements a Velodyne HDL-64E S2 scanner. This sensor can detect objects with a (diffuse) reflectivity of 10% ($= r_{lower}$) at a distance of 50 meter ($= d_{lower}$) and objects with a (diffuse) reflectivity of 80% ($= r_{upper}$) at a distance of 120 meter ($= d_{upper}$). As already mentioned, the amount of light reflected back to the sensor depends on the distance of the object. The decrease in reflected light is compensated within the scanner electronic by lowering the threshold during the scan interval. Unfortunately, this process can not be correctly reproduced by BlenSor, since the information about threshold adaption is not available from the manufacturer. It is however possible to approximate this process by means of linear interpolation of the minimum required reflectivity. We use the 10% and 80% marks listed in the data sheet of the sensor. Objects closer than 50 meter are detected as long as their reflectivity is $> 0\%$. Objects at a distance ($dist$) between 50 meter and 120 meter are detected if their reflectivity is $\geqq r_{min}(dist)$, according to Eq. (1). These values can be easily adapted by the user if an empiric evaluation of the sensor provides different results than the information from the manufacturer. Or if the user wants to simulate a different environmennt like haze or fog. As this effect is calculated on a per-ray basis, it is even possible that a single object is only partially visible if it has a low reflectivity and is far away from the scanner (cf. Fig. 2b).

$$r_{min}(dist) = r_{lower} + \frac{(r_{upper} - r_{lower}) \cdot dist}{d_{upper} - d_{lower}} \tag{1}$$

Once all rays have been cast, we have to impose sensor specific errors to the *clean* measurements ($dist_{real}$). Our error model currently consists of two parts: first, a distance bias ($noise_{bias}$) for each of the 64 laser units. This bias remains the same in each rotation but the noise characteristics can be changed by the user. Experiments with a real Velodyne HDL-64E S2 revealed that the reported

z-distance of a plane normal to the laser's z-axis may differ up to 12 centimeter for any two laser units (combination of a laser and a detector). This is close to the actual numbers provided in the sensor fact sheets. The second part of our error model accounts for the fact that each single measurement ($dist_{noisy}$) is subject to a certain noise as well. Thus a per-ray noise ($noise_{ray}$) is applied to the distance measurements. The final (noisy) distance is formally given by

$$dist_{noisy}(yaw, pitch_i) = dist_{real}(yaw, pitch_i) + \epsilon_{bias,i} + \epsilon_{ray} \tag{2}$$

with $\epsilon_{bias,i} \sim \mathcal{N}(0, \sigma_{bias})$ and $\epsilon_{ray} \sim \mathcal{N}(0, \sigma_{ray})$, where $\mathcal{N}(\mu, \sigma)$ denotes a Normal distribution with mean μ and variance σ.

3.3 Line LIDAR

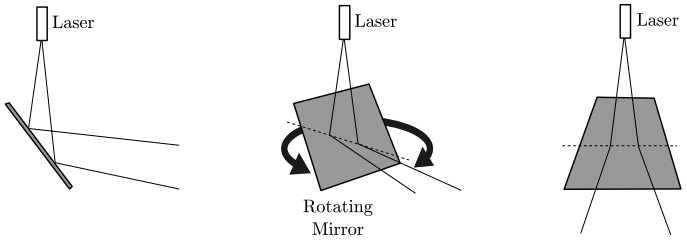
As representative for the Line LIDAR type sensors BlenSor implements a hybrid scanner that can be best described as a combination of an *Ibeo LUX* and a *SICK LMS* sensor with a few modifications. According to the fact sheet of the Ibeo LUX sensor it can detect obstacles with a (diffuse) reflectivity of 10% up to 50 meter and has an average scanning distance of about 200 meter.

The basic principle of measuring distances is described in Section 3.2. A Line LIDAR, however, implements a slightly different method to direct the rays. In contrast to the Velodyne HDL-64E S2 scanner, the line scanner has fixed laser emitters which fire at a rotating mirror. Depending on the position angle of the mirror, the rays are reflected in different directions. The measurement itself is the same as most other laser-based time of flight distance measurement systems. We highlight the fact that the rotating mirror does not only affect the yaw angle of the laser beams but also the pitch angle.

In its initial position (i.e. yaw is 0°) the mirror reflects the rays at the same yaw angle and with the same pitch angle between the rays as they are emitted by the lasers (cf. Fig. 3a). When the yaw angle of the mirror is in the range $[0°, 90°]$, the rays have a yaw and pitch angle which is different from the angles when emitted by the lasers (cf. Fig. 3b). Finally, when the mirror reaches a yaw angle of 90°, the pitch angle of all lasers becomes the same. The former pitch angle between the lasers has become the yaw angle between the lasers (cf. Fig. 3c). The noise model for the measurements is the same as in Section 3.2 due to the same scanning principle.

3.4 Time-of-Flight (ToF) Camera

In contrast to the LIDAR sensors of Sections 3.2 and 3.3, a ToF camera does not need a narrow focused beam of light for its measurements. Consequently, ToF cameras do not use lasers to emit the light pulse. Instead, the whole scene is illuminated at once and the *Time-of-Flight* is measured with a special type of imaging sensor. Compared to the LIDAR sensors, a ToF camera has the advantage of a substantial increase in resolution, however, at the cost of limited measurement distance. In terms of simulation, a ToF camera does not differ much from the other sensors, though. The sensor has a per-ray noise but a



(a) $\alpha = 0°$ **(b)** $\alpha \in [0°, 90°)$ **(c)** $\alpha = 90°$

Fig. 3: The pitch and yaw angle of the outgoing rays is affected by the different yaw angle α of the mirror as it rotates. Only in the mirror's initial position, the angles of the rays are not affected.

higher angular resolution. While LIDAR sensors take a full scanning cycle (i.e. rotation) until they scan the same part of the environment again, subsequent scans of a ToF camera scan the same part of the environment. This may lead to ambiguities in the distance measurements. A signal from one scan may be received in the subsequent scan causing a wrong result. This effect is called *Backfolding*: objects at a certain distance may appear closer than they really are (cf. Fig. 2a). Backfolding can be enabled in BlenSor which causes all distance measurements in the upper half of the maximum scanning distance to be mapped into the lower half according to

$$dist_{backfolding} = \begin{cases} dist_{real}, & dist_{real} < \frac{maxdistance}{2} \\ dist_{real} - \frac{maxdistance}{2}, & else. \end{cases} \quad (3)$$

3.5 Reflection

A special property of all supported sensor types is the total reflection of rays. If a ray hits a reflecting surface it does not immediately produce a measurement. Instead, the ray is reflected at the intersection point with the object and may hit another object at a certain distance. The ray might get reflected again, or not hit an object within the maximum scanning range. Figure 2c illustrates the case when several rays reflected from an object hit another object with a reflectivity above the necessary measurement threshold. As a result, the measured points appear farther away than the object because the rays did actually travel a greater distance. The sensor, however, does not *know* this fact and consequently projects a virtual object behind the real one.

3.6 Ground Truth

An important advantage of BlenSor is the ease at which the ground truth for the simulated scenes can be generated. BlenSor currently supports two output possibilities:

1. The information about the real distance of a ray and the object identifier of the hit object is stored along with the clean & noisy real world data. Every measurement consist of 12 data fields. The `timestamp` of the measurement, `yaw` and `pitch` angle, the measured `distance`, the `noisy distance`, the x, y and z coordinates of the measured points (i.e. *clean* data), the coordinates of the noisy points and the `objectID` of the object that was hit.
2. BlenSor extends the Blender functionality to facilitate exporting of a floating point depth map, rendered at an arbitrary resolution. This depth map can then be used as a ground truth for many algorithms that work on 2.5D data, such as the work of Dolson et al. [1] for instance.

4 Building a simulation

To build a static or dynamic scene for sensor simulation, we can rely on the standard tools of Blender. Any object can be added to the simulation and objects can be imported from other `.blend` files. This resembles the situation of a 3-D modeling artist building a scenery. Technically, there is no limit on the level of scene detail (except RAM of course), but too much detail will result in considerable simulation times. Some material properties (for example the *diffuse reflection* parameter) have an impact on the sensor simulation. The materials can be distributed through `.blend` files and we already made some available on the BlenSor website. This enables other researchers to reuse the materials in their own simulations. In BlenSor, the cameras are the placeholders for the actual sensor devices. Once the scene has been modeled and animated, the user selects a camera that is going *impersonate* the sensor, adjusts its physical properties and eventually simulates the scanning process. No editing of configuration files or any manipulation of scripts is necessary. The simulation is started and configured directly from the camera settings panel. If the simulation is run in *single scan* mode the user has the option to add the ground truth and/or the noisy real world data to the scene (cf. Fig. 1b). This allows for a direct visual verification of the simulation. The scene can be easily adjusted and scanned again. Different scans can coexist in BlenSor, thus allowing a direct comparison of different sensor parameters as well as the scene itself.

4.1 Using the Physics Engine

Physics simulation is possible through the internal physics engine of Blender. BlenSor can simulate any scene that can also be rendered. In order to simulate physical processes, we just need to set up the physics simulation and record the animation data while the physics simulation is running. This has the advantage that the physics simulation needs to be run only once, while the actual sensor simulation can be run as many times as necessary without the need to recalculate the physics.

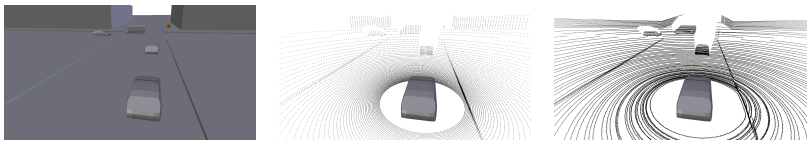
4.2 Exporting Motion Data

To facilitate quantitative analysis of algorithms it is necessary to know the exact position and orientation of all (or at least several) objects in the simulation.

The data of the objects can be exported as a text file describing the state of an object over the scan interval. The user can choose between exporting all, or only a selection of the objects in the scene. Exporting only selected objects may be beneficial for large and complex scenes. To export only selected objects the user literally selects one or more objects within Blender and calls the *Export Motion Data* functionality which was added by BlenSor.

5 Experimental Results

Our first experimental results in Fig. 4a show a crossing scene with four cars. The car closest to the camera is also the position of the sensor. To demonstrate the strength of BlenSor, we use the Velodyne HDL-64E S2 sensor to scan the scene. Figure 4b shows the scene scanned with MORSE, Fig. 4c shows the scene scanned with BlenSor. Compared to the BlenSor results, it is clearly visible that MORSE uses only a rudimentary simulation of the sensor. As a matter of fact, this is no real surprise since the primary focus of MORSE is on realtime simulation of whole robots and less on accurate simulation of sensors with all their properties. The BlenSor scan in contrast shows a much denser scan and a noise level similar to what we would expect with a real Velodyne HDL-64E S2 sensor. It is also important to note that the pitch angle of the laser sensors used by Velodyne is not evenly spaced. Relying on an exemplary calibration file provided by Velodnye, we distribute the pitch angles correctly.



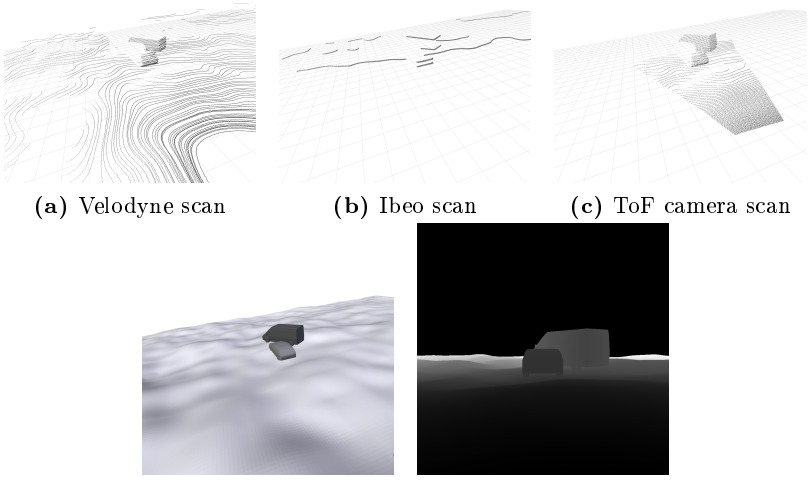
(a) Rendered scene **(b)** Sim. using MORSE **(c)** Sim. using BlenSor

Fig. 4: Simulation of a simple scene with MORSE and BlenSor using the implemented Velodyne HDL-64E S2 sensor.

In our second experiment, illustrated in Fig. 5, we scan a fairly complex scene with 237000 vertices. The terrain has been modified by a displacement map to resemble an uneven surface (e.g. acre). Even though the scene is quite complex, the scanning time for a single simulation interval (in this case $40ms$) is still between 4.9 and 12.8 seconds (see Table 1 for details). Scanning was done on a Intel Core i5 2.53Ghz machine with 3 GB of RAM running a Linux 2.6.31-14 kernel. The average memory usage over the scan is 228 MB.

5.1 Reproducibility

One of the key motivations of developing BlenSor was to allow full reproducibility of research results. BlenSor stores all sensor settings in a `.blend` file. Further, the



(a) Velodyne scan **(b)** Ibeo scan **(c)** ToF camera scan

(d) Rendered scene **(e)** Ground truth

Fig. 5: Simulation of a scene with a large amount of vertices. The scene consists of a rough terrain, simulating an acre, with a near collision of two cars. The figures in the top row show the simulated sensor output of BlenSor, the figures in the bottom row show the rendered scene (i.e. the *camera view*) as well as the ground truth (i.e. a 2000×2000 high-resolution depth map).

Table 1: Processing time in seconds of different sensors in a complex scene.

Velodyne	Ibeo LUX	Time-of-Flight	Depthmap
8.462 [s]	4.943 [s]	5.290 [s]	11.721 [s]

raw scan data can be provided as well in order to allow other researchers to make comparative studies without having to run the simulation again. Nevertheless, storing all needed information in one compact file makes it extremely easy to share the simulation setup. It further enables other researchers to easily modify, adapt or extend the scenarios.

5.2 Scalability

Although sensor simulation is usually a resource intensive task, smaller scenes are rendered almost in realtime by BlenSor. Larger and/or more complex scenes may require substantially more processing time, though. To cope with that problem, BlenSor is designed to allow distribution of the `.blend` file to multiple hosts by splitting the simulated time interval into corresponding sub-intervals. Since the parts are non-overlapping, each host (or thread) can work on its specific sub-interval. Since we do not make use of GPU processing power (which is usually the

case for simulators that rely on a a game engine), we can run several instances of simulation on a multi-core machine at the same time as well.

6 Conclusion

In this article we introduce a software tool for reproducible research in range data processing. Due to the strong linkage among simulation and modeling, creation of ground truth data is very simple. In fact, BlenSor considerably simplifies simulation of otherwise untestable scenarios (e.g. crashes). At the time of writing, all implemented sensor types already produce data that closely resembles the output of real sensors. We hope that this software encourages reproducible research in the respective fields and simplifies the distribution of test data for comparative studies. There is also good reason to believe that the functionality of BlenSor allows more researchers to develop algorithms for range scanner data without having to possess the physical sensor. Future work on BlenSor will also include support for the mixed-pixel error ([5,6]), refraction and, of course, additional sensors (i.e Hokyuo and SICK sensors).

References

1. J. Dolson, J. Baek, C. Plagemann, and S. Thrun. Upsampling range data in dynamic environments. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR '10)*, pages 1141–1148, San Francisco, CA, USA, 2010.
2. G. Echeverria, N. Lassabe, A. Degroote, and S. Lemaign. Modular open robots simulation engine: Morse. In *Proceedings of the IEEE Conference on Robotics and Automation (ICRA '10)*, Shanghai, China, 2011.
3. M. Kretkiewicz J. Bedkowski and A. Mastowski. 3D laser range finder simulation based on rotated LMS SICK 200. In *Proceedings of the EURON/IARP International Workshop on Robotics for Risky Interventions and Surveillance of the Environment*, Benicassim, Spain, January 2008.
4. D. Meissner and K. Dietmayer. Simulation and calibration of infrastructure based laser scanner networks at intersections. In *Proceedings of the IEEE Intelligent Vehicles Symposium (IV '10)*, pages 670 – 675, San Diego, CA, USA, 2010.
5. D. Huber P. Tang and B. Akinci. A comparative analysis of depth-discontinuity and mixed-pixel detection algorithms. pages 29–38, Los Alamitos, CA, USA, 2007.
6. E. Gregorio-Lopez R. Sanz-Cortiella, J. Llorens-Calveras J.R. Rosell-Polo and J. Palacin-Roca. Characterisation of the LMS200 laser beam under the influence of blockage surfaces. influence on 3D scanning of tree orchards. *Sensors*, 11(3):2751–2772, 2011.
7. R. Vaughan. Massively multi-robot simulation in stage. *Swarm Intelligence*, 2(2):189–208, December 2008.