Automatic Generation of Random Step Environment Models for Gazebo Simulator

Ruslan Gabdrahmanov[1], Tatyana Tsoy[1], Yang Bai[2], Mikhail Svinin[2], and Evgeni Magid[1](✉)

[1] Laboratory of Intelligent Robotic Systems, Intelligent Robotics Department, Kazan Federal University, 35 Kremlevskya Street, Kazan, Tatarstan Republic, Russian Federation
ruslan3452364@yandex.ru, tt@it.kfu.ru dr.e.magid@ieee.org

[2] Information Science and Engineering Department, College of Information Science and Engineering, Ritsumeikan University, 1-1-1 Noji-higashi, Kusatsu, Shiga 525-8577, Japan
{yangbai,svinin}@fc.ritsumei.ac.jp
http://robot.kpfu.ru/eng

Abstract. Negotiating rough terrain obstacles is a key task of a mobile terrestrial robot. One of the most popular standards in rough terrain modeling is the NIST Random Step Environment (RSE) or Random Stepfield, which allows constructing a broad variety of debris-like environments. This paper proposes a virtual RSE models' automatic generator LIRS-RSEGen for the Gazebo simulator. We analyzed typical for RSEs obstacles and structures, determined parameters that are required in order to generate a RSE and implemented a generator, which constructs RSE Gazebo worlds that could be further edited or directly used within the Gazebo. Constructed by our generator worlds were validated in the Gazebo using virtual models of TurtleBot3 wheeled robot and Servosila Engineer crawler robot.

Keywords: USAR · UGV · Modelling · Gazebo · Random step environment

1 Introduction

An unmanned ground vehicle (UGV) is a mobile robot that locomotes on a supporting surface of various types, including rough terrain. A UGV is the most broadly used type of robots within a large variety of tasks, including Urban Search and Rescue (USAR) robotics. A typical operational environment of USAR contains heaps of trash and debris, formed by construction materials, pieces of furniture, various equipment, home and office objects that complicate vision, locomotion and SLAM applications [10,16].

Simulations are used in many areas of robotics, including design and construction of a robot, development of control algorithms, offline programming,

D. Chugo et al. (Eds.): CLAWAR 2021, LNNS 324, pp. 408–420, 2022.
https://doi.org/10.1007/978-3-030-86294-7_36

demonstrations, etc. [24]. They allow creating a virtual model of a robot and an environment [20,21], and a behavior of a virtual model attempts to approximate a real robot and a real world environment as precise as possible [3]. Simulations in robotics are widely used for multiple reasons, including cost of virtual errors vs real world ones [17] and ease of test cases reproducing [22].

For our research we selected the Gazebo simulator [4] because of its compatibility with robot operating system (ROS [15]) and high-quality physics. There exist several tools for generating worlds in Gazebo, e.g., an automatic tool for Gazebo world construction LIRS-WCT [1] that creates a 3D world from a 2D gray-scale image. LIRS-WCT tool is easy to use and saves significant time for constructing various mazes and rugged terrains. A similar tool was presented in [8], but it had issues with a low real time factor (RTF) of the Gazebo simulation due to the nature of objects' models, which required the physics engine to perform a large number of self-collision checks.

While both tools [1,8] could be successfully employed to create a 3D RSE, which is described in details in Sect. 2, they would ignore a simplicity of a RSE model and require a 2D image (for both tools) and a texture (for LIRS-WCT tool) as an input. The new tool focuses on constructing 3D RSE worlds with a minimal negative impact on the Gazebo RTF, CPU load and memory usage, and does not require a prearranged 2D image input. The tool was with the TurtleBot3 [2] and Servosila Engineer [12] UGVs, and created by the tool RSE environments demonstrated good performance.

2 Random Step Environment or Random Stepfield

The Random Step Environment (RSE) or Random Stepfield is a simulation of a cluttered debris environment proposed by the National Institute of Standards and Technology to evaluate the performance of USAR robots [5]. RSEs are actively used for testing of algorithms related to overcoming rugged terrain. In [9] the authors presented a classification of measuring a static balance of a two-crawler robot and employed a RSE to simulate a rough terrain with the definitions of pose types that strongly corresponded to the robot posture on the RSE. In [14] the authors performed robot simulation physics validation while having RSE as one of the main elements of their test sites. In [18] the authors proposed an algorithm for extraction of terrain features from range images. A RSE with relatively small elevation differences was used for development and testing of a machine learning based algorithm that substituted conventional controllers in obstacles' overcoming task [19]. Naturally, RSEs serve as mobility and autonomy test fields in RoboCup Rescue Robot League competitions [6], where at least several arenas of different complexity are constructed in a form of a RSE.

A single RSE patch is a wooden frame with an external boundary that contains a 10×10 size matrix. Each cell hosts a wooden block that forms a single RSE "step". While block sizes might slightly vary, one of the typical conventions (used by our research groups of Kazan Federal University and Ritsumeikan University) employs 10 cm width and 10 cm depth blocks of a height H

$\in \{0, 10, 20, 30, 40\}$ cm. The external boundary is 10 cm height, 10 cm depth and 120 cm width; since each of two adjacent blocks has a boundary, together they always form a double-boundary in-between standard patches. Mixing these steps in different combinations allows creating a broad variety of rough terrain testing grounds, which can form typical for real world stable debris structures [14] that may contain horizontal barriers (Fig. 1), diagonal barriers (Fig. 2), traversable or non-traversable peaks (Fig. 3), or completely random profiles of a terrain (Fig. 4). In RoboCup Rescue competitions RSEs might deviate from keeping the standard $\{0, 10, 20, 30, 40\}$ cm heights as well as from having a 10×10 blocks patches [6].

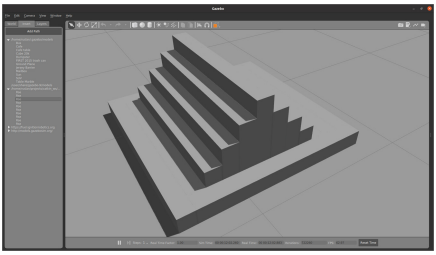

Fig. 1. A single horizontal barrier.

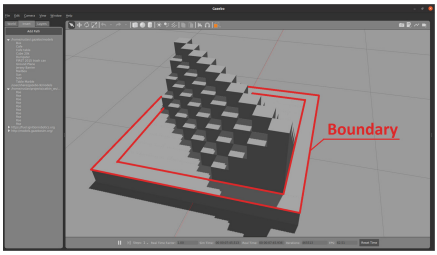

Fig. 2. A single diagonal barrier.

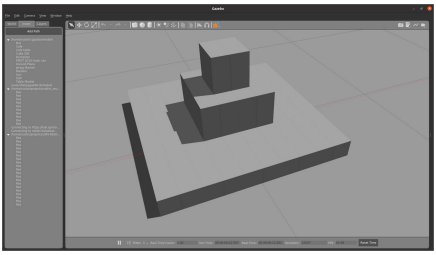

Fig. 3. A single non-traversable peak.

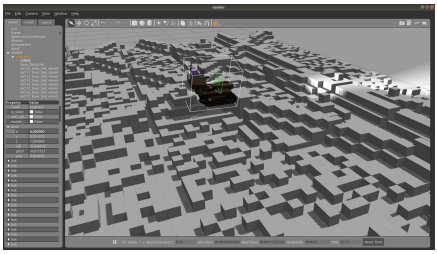

Fig. 4. A large-size random profile RSE.

As a simulation of debris, RSE has a number of advantages, which include ease of construction, availability and affordable costs of construction materials, a high mobility and ease of storage of a testing field, a broad variety of possible forms of an environment, and a possibility to rapidly rearrange the environment in a real time in a course of an experiment. Moreover, since the construction is modular, any damaged or destroyed block ("step") could be easily replaced. From a virtual simulation point of view, since RSE has a small number of polygons that form at most 5 visible facets for each block, high performance and low memory requirements could be achieved (if implemented correctly). Disadvantages of the

RSE approach include inability to construct some rather typical for real world structures and obstacles (e.g., an inclined plane) and a simplicity of the model compared to the real USAR environment [7]. These issues decrease the efficiency of RSE-based testing of new approaches as a further transition to a real world application might demonstrate an unexpectedly significant under-performance.

3 RSE Generation Tool

We studied various existing approaches of a RSE construction in terms of RSE structure, identified typical obstacles, their formation on a RSE pallet and parameters that affect a shape, a height and other characteristics of these obstacles and RSE as an integral structure. We developed an algorithm that receives a user-defined desired distribution of RSE blocks and settings to form a 2D matrix, which represents heights of blocks in each entry of the matrix. Next, a versatile module constructs a 3D RSE model from the matrix. Two basic 3D model formation modules were created. The first module forms a single optimized model of the .obj format from the matrix, which could be further imported as a standard Gazebo model. The second module generates a Gazebo world in which each RSE block acts as a separate independent model. The later approach produces a world, which is significantly less productive, but it allows further manual editing of the world directly in the Gazebo.

The created modules were encapsulated in a single generator, named LIRS-RSEGen, with a simple graphical user interface GUI (Fig. 5). The generator was tested in various modes and settings, while evaluating suitability of constructed models for testing rough terrain navigation and obstacle negotiation algorithms for UGVs within typical USAR scenes. An impact of the created models performance of the simulation was evaluated using RTF, FPS, system CPU and GPU load and other qualitative parameters.

The generator GUI has a large number of settings (Fig. 5) to determine two types of parameters. The first six listed below parameters have a global nature and are set for the entire RSE world (for all obstacles). The rest of the parameters have a local nature and are set for each obstacle independently. In the following list the parameters are arranged in order of their appearance, from the top toward the bottom of the GUI, from left to right:

1. Number of maps to generate: a number of generated RSE Gazebo worlds.
2. Single chunk size: a length of a single chunk edge L_C in RSE blocks. Each chunk is a $L_C \times L_C$ square.
3. Map size in chunks: a length of a map edge M_C in chunks, i.e., currently only square maps could be generated. For example, a selection of $M_C = 2$ and $L_C = 10$ generates a map of 20×20 RSE cells size (or 24×24 including boundaries if *Chunk boundary* parameter is set to "yes"; in this case two rows and columns correspond to outer boundaries of the entire RSE and two others - to the inner double-boundary between two adjacent chunks).
4. Height discretization step: a height of a single height step H_{DS} in cm. For example, a selection of *Max obstacle height* as 5 units and $H_{DS} = 10$ cm give possible heights of the RSE blocks within $\{0, 10, 20, 30, 40\}$ cm set.

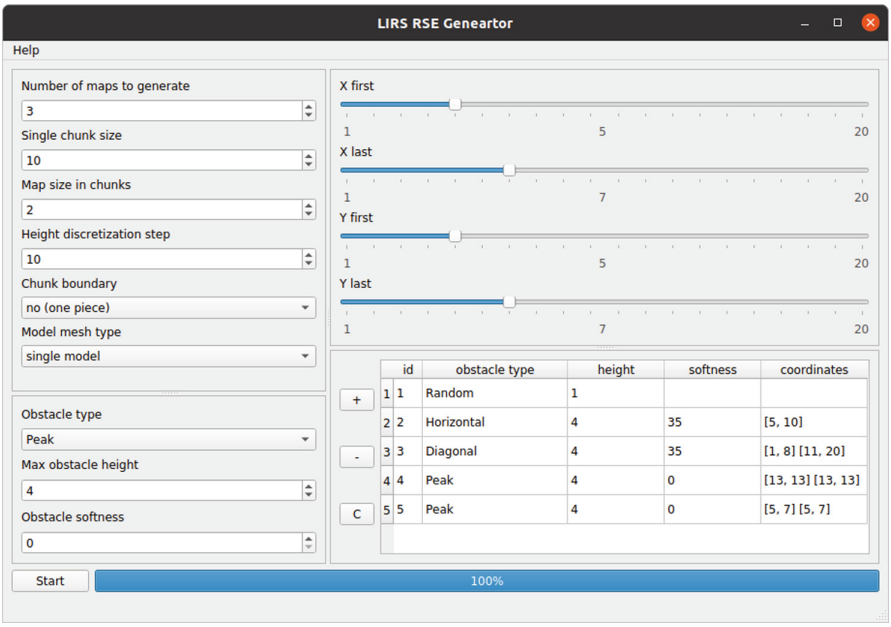

Fig. 5. GUI of LIRS-RSEGen tool with a peak positioning input settings, which correspond to ID 5 obstacle in the table (in the right bottom corner of the GUI).

5. Chunk boundary: defines whether to create a single external boundary and double internal boundaries between chunks (value "yes(divided)") or not (value "no"). Figures 1–3 demonstrate examples of 10×10 RSE cells' chunks; while for Fig. 1 and Fig. 2 boundaries are selected and this automatically increases the chunk total size to 12×12 RSE cells (the one-cell boundary is highlighted in red color in Fig. 2), Fig. 3 shows a peak without boundaries of a chunk keeping the original 10×10 RSE cells' size of the chunk.
6. Model mesh type: a "single model" creates a single large model within the Gazebo world that contains all blocks of the RSE; a multi model creates a one independent model per each RSE block. The later has a lower performance, but allows easy editing of the world.
7. Obstacle type: a type of a new obstacle pattern, which is a selection from a list {diagonal barrier, horizontal barrier, peak, random, normal random}.
8. Max obstacle height: a maximal height of a generated obstacle in units (where each unit has a height of H_{DS} cm).
9. Obstacle softness: a value s that could be selected in [0..100]; large values create blocks that are closer to a flat environment appearance, while s=0 generates a vertical obstacle. The calculations are performed relatively to the current chunk.
10. X first, Y first, X last and Y last coordinate sliders (the right top side of the GUI) allow setting a position of a new obstacle. For example, for a diagonal barrier obstacle they define vertices that serve as endpoints of the diagonal

barrier, which is embedded in a square with this diagonal. For a peak, they similarly define a square that circumscribes the new peak location. In in order to allow the user verifying the selected values, the selected number appears under a central part of each slider.

After the parameters' settings are completed, the information about the current obstacle is stored within the generator. Next, a new obstacle could be added with "+" button (Fig. 5, central bottom part), and the three local parameters and the four coordinate sliders should be set. The information about all currently scheduled obstacles within the RSE is available in the table, in the bottom right part of the GUI. The table provides obstacle unique ID number, type, height, softness and (X, Y) coordinates. The user can remove a last planned obstacle by pushing − button or remove all obstacles with *C* ("Clear all") button.

Button *Start* in the left bottom corner launches the generator with the planned obstacles, which are generated iteratively in the order of their appearance in the list, and a progress bar to the right of the button shows the progress of the generation process. Finally, button *Help* in the left top corner provides a user with a brief manual for the tool.

The LIRS-RSEGen creates RSE models and worlds suitable for importing into or directly running in the Gazebo simulator. The Gazebo 3D engine for simulating robots and their environment enables working with the simulation world at a run time and allows to recreate and import RSE models on the fly. The generator operates in two stages: the first (a 2D matrix construction) is performed by a block height matrix generator and the second (a Gazebo 3D world construction) is performed by a tool for forming 3D blocks from the given matrix entries. Currently, the matrix generator of LIRS-RSEGen supports five generation modes, each using a different algorithm of a matrix construction:

1. Random - a random distribution of blocks is generated throughout an entire RSE (Fig. 4).
2. Random Gaussian - generates a random Gaussian distribution of blocks throughout the RSE, which is slightly smoother than the Random.
3. Horizontal - a horizontal barrier type obstacle (Fig. 1) that appears between two opposite boundaries (or between two RSE cells that have the same X or the same Y coordinate; these two cells define a central axis of the barrier) of the RSE and follows particular user-provided settings, which define its length, height and shape.
4. Diagonal - a diagonal barrier type obstacle (Fig. 2) that appears between two opposite boundaries of the RSE and follows particular user-provided settings, which define its length, height and shape.
5. Peak - a peak type obstacle (Fig. 3), whose length, height and shape are also defined by a user's particular settings.

At first, the program creates an initial square zero matrix M of K × K size, where K is calculated using a number of user-defined parameters: a selected size of a map edge in chunks, a chunk size and a chunk's division selection (of boundary). A chunk is defined as a single square RSE patch of N × N size,

where N is a single chunk size. Each chunk is separated from other chunks by a 10 cm height double-boundary if the chunk boundary parameter is set as "yes (divided)" or it has no boundary if the chunk boundary parameter is set to "no". For example, if a user selects a map without boundaries, Map A with $size\_in\_chunks = 2$ and $chunk\_size = 10$ will form exactly the same initial zero matrix $M_{20\times 20}$ as Map B with $size\_in\_chunks = 1$ and $chunk\_size = 20$.

A user specifies obstacles one by one to form a list of obstacles for the Gazebo world; the list appears as a table in the right bottom part of the GUI (Fig. 5). The obstacles are imposed on the initial matrix iteratively in the order they were added to the list. This procedure could cause a situation where an obstacle that is added at time $t + 1$ could rearrange an obstacle that has been added in time t if they are co-located. This is a purposeful feature of the LIRS-RSEGen.

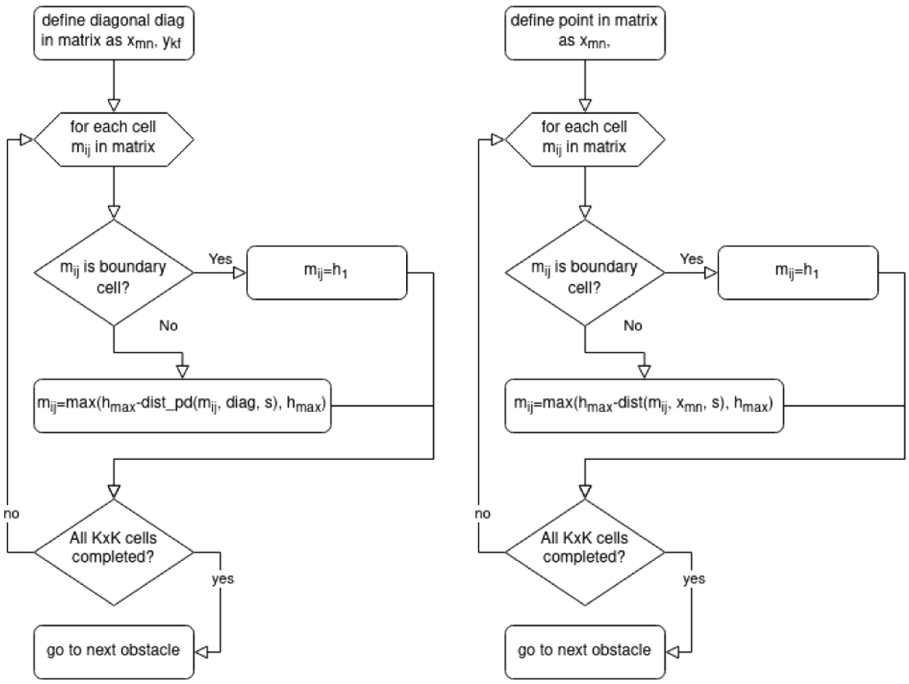

Fig. 6. Diagonal barrier (left) and peak (right) obstacle matrix generation algorithms.

Figure 6 presents algorithms of a diagonal (left) and a peak obstacle construction (right). Value K defines a size of matrix $M_{K\times K}$ (an entire RSE) with entries m_{ij}, where $i, j \in [0, K-1]$. The array *diag* contains two endpoints x_{mn} and y_{kf} (where m, k denote columns, and n, f denote rows of matrix M) that define a random diagonal line between two opposite boundaries of the entire RSE. Function *dist_pd* calculates a shortest distance between m_{ij} and the defined *diag*

line. Function *dist* calculates a distance between m_{ij} coordinate and a peak origin x_{mn}. The parameter h_1 denotes a selected in GUI height discretization step in cm, which is global for the entire RSE; h_{max} denotes a maximum possible height of the obstacle in cm (a multiplication of height discretization step in cm and "Max obstacle height", which a user defines in units independently for each obstacle); s denotes "Obstacle softness", which is a rate of change of a height function for each matrix entry that reflects changes in height with distance from the diagonal (or the peak origin respectfully). Function max(a, b) returns a maximum value of the two input values a and b.

The second stage of the generation forms the Gazebo world of 3D blocks from the constructed 2D matrix. The tool works differently depending on the setting of the Model mesh type. The single model selection transforms the matrix, which was created on the first stage, into a 3D RSE model and creates an empty world with this single model. The multi model selection creates a world where each block is a separate model. The multi model converter works quite simply: after receiving height and position values, it creates new box models of the specified height in the specified locations and stores it in the Gazebo world.

The single model mode is more complicated and works directly with .obj models. This converter creates a set of 8 vertices and 6 polygons for each block, constructing a block of a desired height at a given position and applies a single set of vertex normals to all blocks. After the .obj file is generated, all additional files and structures are generated, such as the .mtl file and others. This way, the resulting model can be seamlessly imported directly into any Gazebo world or into a running simulation on a fly and additionally an empty world with the generated model will be created.

The generator was implemented in python 3.6 language, using *numpy* [13] and *PyQt*5 [23] libraries. The LIRS-RSEGen follows the SDF model standard for a world generation and the OBJ geometry file format for 3D models.

4 Validation of Virtual RSE Models

Two robot models were used for validating of virtual RSE models: Servosila Engineer and Turtlebot3 Burger. The Servosila Engineer robot is a crawler UGV designed by Russian company Servosila for operating in rough terrain environments (Fig. 7, left). The robot has a five DoF arm with a gripper, two main tracks and two additional flippers with an adjustable angle, which are operated by a single motor. On-board sensors include a laser range finder, three front RGB cameras and a rear RGB camera. While visually the model is almost an exact replica of the real robot and its main physical characteristics well correspond to the real ones [11], the model causes a quite high load on a PC [12] (Table 1 presents the PC specifications); the RTF in an empty Gazebo world with a static robot model was 0.28–0.3 on average.

The Turtlebot3 Burger is a simple differential drive UGV of a modular construction, popular in education and research (Fig. 8). It consist of a base, two powered wheels, a Raspberry PI microcomputer, an IMU and a lidar. Due to its low cost and ease of operation, the robot is well suited for preliminary tests.

Table 1. Testing computer specifications

Module	Model	Characteristics
CPU	AMD Ryzen 7 2700 X	8 × 3.70 GHz
GPU	Nvidia GeForce1660	6 GB 1830 MHz
RAM	-	16 GB

To evaluate the generator's performance, a single obstacle Gazebo world was constructed, where an obstacle was a small (single chunk with $L_C = 10$) RSE of each type of obstacles (Fig. 1–3). Next, a number of more complicated and large-size obstacles were constructed, including 20 × 20, 40 × 40, 60 × 60 and 80 × 80 size RSEs with $H_{DS} = 10$ cm and each obstacle height of up to 5 or 10 units. The Servosila Engineer robot was placed into the constructed worlds and the system load was studied using such parameters as RTF, CPU load and memory load. We are interested in minimizing CPU load and memory load while maximizing RTF: the closer this parameter to one, the faster is the simulation speed within the Gazebo. The recommended minimally acceptable RTF level for comfortable use of the Gazebo simulator should be at least 0.3 [1]. Since the obtained with the Servosila Engineer robot RTFs were between 0.24 and 0.3, in order to eliminate the possible negative influence of a non-optimal modelling (having 0.27–0.3 RTF in the empty world clearly hinted on this issue) it was decided to validate the generated worlds with other robots as well.

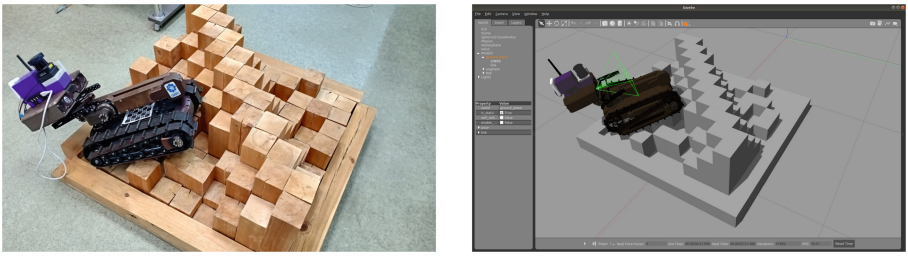

Fig. 7. The Servosila Engineer traverses RSE diagonal barrier in a real world at LIRS (left) and in the Gazebo (right). $L_C = 10$, boundaries are enabled, $H_{DS} = 10$ cm.

Three TurtleBot3 Burger robots were spawned in the same type of RSE worlds with $H_{DS} = 1$ cm, which enabled the small wheels of the robots to negotiate the RSE blocks (Fig. 8). The obtained RTF stayed within 0.92–1 being accompanied with low CPU and memory load. Table 2 summarizes results of all virtual experiments: CPU and memory load values were averaged for three experiments for random paths; RTF value stored a minimal and a maximal registered level that were obtained while monitoring the UGVs locomotion along its path. Column "RSE max height" contains measurements of maximal obstacle

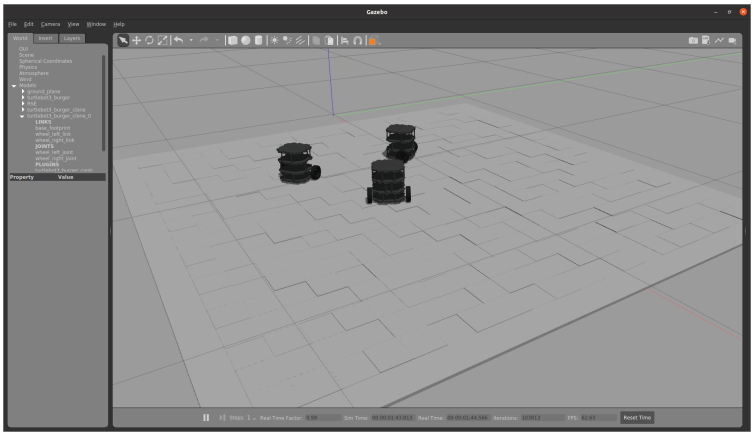

Fig. 8. The three Turtlebot3 robots on a RSE with height $H_{DS} = 1$ cm in the Gazebo.

Table 2. Performance evaluation with various size and height settings

Robot Model × number	RSE Size $L_C \times L_C$	RSE max height Units	RTF Min-Max	CPU load %	Memory load Gb
Engineer	Empty world	N/A	0.27–0.30	26%	7.6
Engineer	10 × 10	5	0.26–0.29	26%	7.6
Engineer	20 × 20	5	0.25–0.28	26%	7.6
Engineer	40 × 40	10	0.25–0.28	26%	7.7
Engineer	60 × 60	10	0.25–0.28	26%	7.7
Engineer	80 × 80	10	0.24–0.28	26%	7.7
Turtlebot 3 × 3	Empty world	N/A	0.98–1	8%	0.8
Turtlebot 3 × 3	10 × 10	5	0.97–1	8%	0.8
Turtlebot 3 × 3	20 × 20	5	0.97–1	8%	0.8
Turtlebot 3 × 3	40 × 40	5	0.96–1	8%	0.8
Turtlebot 3 × 3	60 × 60	5	0.95–1	8%	0.8
Turtlebot 3 × 3	80 × 80	5	0.92–1	8%	0.8

height in units while the unit height is different for the different robots: $H_{DS} = 10$ cm for the Servosila Engineer and $H_{DS} = 1$ cm for the TurtleBot3 Burgers. The virtual experiments demonstrated a rather low impact of the RSE size on the performance. The impact was clearly detected only for the RTF parameter for both types of robots and in the memory load for the Servosila Engineer robot.

We run several experiments in teleoperation mode to repeat virtual Servosila Engineer robot's locomotion within a constructed by the generator RSE. Figure 7 demonstrates that it was possible to construct exactly the same real

world RSE at the Laboratory of Intelligent Robotic Systems (LIRS)[1] using its virtual Gazebo model as a source and to employ it for the robot locomotion tests.

5 Conclusions

This paper presented an automatic generator of a virtual Random Step Environment (RSE) models for the Gazebo simulator that allows constructing typical for RSEs obstacles as well as non-standard user-defined models. A convenient GUI enables selecting various parameters of RSE model, including map size, maximal height of obstacles, heights' sampling, softness and model mesh type, which permits to construct the entire RSE as a single model within the Gazebo world (improves a performance) or to generate separate models for each block of the RSE (allows a further model adjustment at a run time).

Constructed by LIRS-RSEGen worlds were validated in the Gazebo simulator using a virtual model of the Servosila Engineer crawler robot and three simultaneously running wheeled TurtleBot3 UGVs. Tests demonstrated effectiveness of constructed RSE worlds in terms of the Gazebo real time factor (RTF), CPU and memory load, which had acceptable values even for a relatively large RSEs of 80×80 block size. The tool is available for free academic use at Gitlab account of our Laboratory of Intelligent Robotic Systems (LIRS)[2].

Acknowledgment. This work was supported by the Russian Foundation for Basic Research (RFBR), project ID 19–58-70002. The third and forth authors acknowledge the support of the Japan Science and Technology Agency, the JST Strategic International Collaborative Research Program, Project No. 18065977.

References

1. Abbyasov, B., Lavrenov, R., Zakiev, A., Yakovlev, K., Svinin, M., Magid, E.: Automatic tool for gazebo world construction: from a grayscale image to a 3D solid model. In: 2020 IEEE International Conference on Robotics and Automation (ICRA), pp. 7226–7232. IEEE (2020)
2. Amsters, R., Slaets, P.: Turtlebot 3 as a robotics education platform. In: International Conference on Robotics in Education (RiE), pp. 170–181. Springer (2019). https://doi.org/10.1007/978-3-030-26945-6_16
3. Borisov, A., Kuznetsov, S., Mamaev, I., Tenenev, V.: Describing the motion of a body with an elliptical cross section in a viscous uncompressible fluid by model equations reconstructed from data processing. Tech. Phys. Lett. **42**(9), 886–890 (2016)
4. OSR Foundation: Gazebo official site (2021). http://gazebosim.org/
5. Jacoff, A., Downs, A., Virts, A., Messina, E.: Stepfield pallets: repeatable terrain for evaluating robot mobility. In: Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems, pp. 29–34 (2008)

[1] https://kpfu.ru/eng/itis/research/laboratory-of-intelligent-robotic-systems.
[2] LIRS-RSEGen, GitLab, https://gitlab.com/LIRS_Projects/LIRS-RSEGen.

6. Jacoff, A., et al.: Using competitions to advance the development of standard test methods for response robots. In: Proceedings of the Workshop on Performance Metrics for Intelligent Systems, pp. 182–189 (2012)
7. Jakobi, N., Husbands, P., Harvey, I.: Noise and the reality gap: the use of simulation in evolutionary robotics. In: European Conference on Artificial Life, pp. 704–720. Springer (1995). https://doi.org/10.1007/3-540-59496-5_337
8. Lavrenov, R., Zakiev, A.: Tool for 3D gazebo map construction from arbitrary images and laser scans. In: 2017 10th International Conference on Developments in eSystems Engineering (DeSE), pp. 256–261. IEEE (2017)
9. Magid, E., Tsubouchi, T.: Static balance for rescue robot navigation: discretizing rotational motion within random step environment. In: International Conference on Simulation, Modeling, and Programming for Autonomous Robots, pp. 423–435. Springer (2010). https://doi.org/10.1007/978-3-642-17319-6_39
10. Malov, D., Edemskii, A., Saveliev, A.: Proactive localization system as a part of a cyberphysical smart environment. In: 2019 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM), pp. 1–5. IEEE (2019)
11. Moskvin, I., Lavrenov, R.: Modeling tracks and controller for servosila engineer robot. In: Proceedings of 14th International Conference on Electromechanics and Robotics "Zavalishin's Readings", pp. 411–422. Springer (2020). https://doi.org/10.1007/978-981-13-9267-2_33
12. Moskvin, I., Lavrenov, R., Magid, E., Svinin, M.: Modelling a crawler robot using wheels as pseudo-tracks: model complexity vs performance. In: 2020 IEEE 7th International Conference on Industrial Engineering and Applications (ICIEA), pp. 1–5. IEEE (2020)
13. Oliphant, T.E.: A guide to NumPy, vol. 1. Trelgol Publishing USA (2006)
14. Pepper, C., Balakirsky, S., Scrapper, C.: Robot simulation physics validation. In: Proceedings of the 2007 Workshop on Performance Metrics for Intelligent Systems, pp. 97–104 (2007)
15. Quigley, M., et al.: Ros: an open-source robot operating system. In: ICRA Workshop on Open Source Software, vol. 3, p. 5. Kobe, Japan (2009)
16. Safin, R., Lavrenov, R., Martínez-García, E.A.: Evaluation of visual slam methods in usar applications using ros/gazebo simulation. In: Proceedings of 15th International Conference on Electromechanics and Robotics "Zavalishin's Readings", pp. 371–382. Springer (2021). https://doi.org/10.1007/978-981-15-5580-0_30
17. Shabalina, K., Sagitov, A., Su, K.L., Hsia, K.H., Magid, E.: Avrora unior car-like robot in gazebo environment. In: International Conference on Artificial Life and Robotics, pp. 116–119 (2019)
18. Sheh, R., Kadous, M., Sammut, C., Hengst, B.: Extracting terrain features from range images for autonomous random stepfield traversal. In: IEEE International Workshop on Safety, Security and Rescue Robotics, Rome (2007)
19. Sheh, R., Hengst, B., Sammut, C.: Behavioural cloning for driving robots over rough terrain. In: 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 732–737. IEEE (2011)
20. Sheh, R., et al.: Advancing the state of urban search and rescue robotics through the robocuprescue robot league competition. In: Field and service robotics, pp. 127–142. Springer (2014). https://doi.org/10.1007/978-3-642-40686-7_9
21. Simakov, N., Lavrenov, R., Zakiev, A., Safin, R., Martínez-García, E.A.: Modeling usar maps for the collection of information on the state of the environment. In: 2019 12th International Conference on Developments in eSystems Engineering (DeSE), pp. 918–923. IEEE (2019)

22. Timperley, C.S., Afzal, A., Katz, D.S., Hernandez, J.M., Le Goues, C.: Crashing simulated planes is cheap: can simulation detect robotics bugs early? In: 2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST), pp. 331–342. IEEE (2018)
23. Willman, J.: Overview of pyqt5. In: Modern PyQt, pp. 1–42. Springer (2021). https://doi.org/10.1007/978-1-4842-6603-8_1
24. Yakovlev, K., Baskin, E., Hramoin, I.: Grid-based angle-constrained path planning. In: Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz), pp. 208–221. Springer (2015). https://doi.org/10.1007/978-3-319-24489-1_16