Tool for 3D Gazebo map construction from arbitrary images and laser scans

Roman Lavrenov
Intelligent Robotics Dept., Higher Institute for Information Technology and Information Systems (ITIS)
Kazan Federal University
Kazan, Russian Federation
e-mail: lavrenov@it.kfu.ru

Aufar Zakiev
Intelligent Robotics Dept., Higher Institute for Information Technology and Information Systems (ITIS)
Kazan Federal University
Kazan, Russian Federation
e-mail: zaufar@gmail.com

***Abstract* — Algorithms for mobile robots, such as navigation, mapping, and SLAM, require proper modelling of environment. Our paper presents an automatic tool that allows creating a realistic 3D-landscape in Gazebo simulation, which is based on the real sensor-based experimental results. The tool provides an occupancy grid map automatic filtering and import to Gazebo framework as a heightmap and enables to configure the settings for created simulation environment. In addition, the tool is capable to create a 3D Gazebo map from any arbitrary image.**

Keywords-Gazebo; ROS; octomap; occupancy grid; heightmap; map filtering;

I. INTRODUCTION

At present, mobile robotics are developing rapidly. Navigation problem is one of the most important difficulties in mobile robotics. The main components of the navigation of an autonomous vehicle are localization, mapping and path planning [1]. Localization is responsible for exact determining of robot's current position in an environment. Mapping process is a collecting sensory data about environment and storing it in a form that is convenient for further using. Path planning is a search of route within an environment from start to target position while utilizing data about obstacles that was collected during a mapping process [17,18]. SLAM unites localization and mapping processes to achieve autonomous operation of the mobile vehicle. SLAM algorithms cope with various particular hardware and/or environmental constraints, e.g. monocular SLAM methods [2], LIDAR and visual sensors combining SLAM methods [3].

Navigation and SLAM algorithms should be carefully tested before integrating them into real mobile robot software. Testing is usually performed with a help of a computer simulation, which is an efficient and inexpensive way to verify new methods, algorithms correctness and possibility to apply them on existing robotic systems [4]. The best way to create simulated environment is to employ real sensor-based input data. For our research project on SLAM for a heterogeneous group using active collaborative vision and multi-robot belief space planning, we are utilizing Robot Operating System (ROS). However, it is currently missing a simple and convenient tool for real sensor-based data import and processing, and such tool development is a core contribution of our paper.

The rest of paper is organized as follows. Section 2 introduces project motivation and system setup. Section 3 overviews filter types and justifies our selection of a best suitable filter, while Section 4 presents details of map importing process. Section 5 shows the example of using the tool. Finally, we summarize our research in section 6.

II. MOTIVATION AND SYSTEM SETUP

A. Project motivation

The main goal of our research deals with collaboration aspects investigation between a heterogonous group of a UGV and a number of small-size UAVs, focusing on operation in uncertain environments. The environment could be represented by an outdated or imprecise map. To facilitate a reliable autonomous operation, robots will collaboratively perceive the surrounding environment and plan high-quality actions while taking different sources of uncertainty into account. We build upon recent progress in SLAM and belief space planning [5], and will explore approaches that take into account the uncertainty in the environment. In particular, such a framework will allow robots to devise proper motion to improve localization and mapping even in lack of GPS signal.

Our first and very simplified team collaboration approach dealt with two quadrotors performing 3D-mapping with Kinect sensors and path planning with a Husky robot [6]. Husky robot employed data from quadrotors and its LIDAR sensor to create Voronoi Graph, performed global path planning and followed a selected path while applying local re-planning for dynamic obstacles avoidance. The simulation was performed in a synthetic ROS/Gazebo environment, and all phases of the operation (namely, 3D mapping, path calculation and locomotion) were very sensitive to landscape changes. To further sophisticate the simulation, we want to substitute a synthetic environment with a real sensor-based map.

This work was partially supported by the Russian Foundation for Basic Research (RFBR) and Ministry of Science Technology & Space State of Israel (joint project ID 15-57-06010). Part of the work was performed according to the Russian Government Program of Competitive Growth of Kazan Federal University.

B. Simulation setup

For our research, we use ROS Indigo (Robot Operating System) together with Gazebo 2.2 simulator that has a built-in physics engine, convenient programmable and graphical interfaces, and allows creating high-quality simulations. We perform mapping with occupancy grid and octomap approaches within ROS/Gazebo environment.

Occupancy grid is a field of values, each representing an obstacle presence in a specified location [7]. Values could be binary (0 for a free cell, 1 for an occupied with an obstacle cell) or store non-binary data that reflects terrain roughness, i.e. particular region traversability [8]. When occupancy level varies from 0 to 255, an occupancy grid is visualized as grayscale image with 0 assigned for a free cell and 255 for a completely non-traversable cell (Fig. 3, both).

Octomap is another way to store information about 3D-space occupancy. To store data effectively, entire volume is divided into voxels, and each voxel state is stored in nodes of a corresponding octree (a tree with a branching factor of 8) [9]. This provides map resolution control with limiting octree depth. An example of voxel structure and its octree representation are demonstrated in Fig. 1.

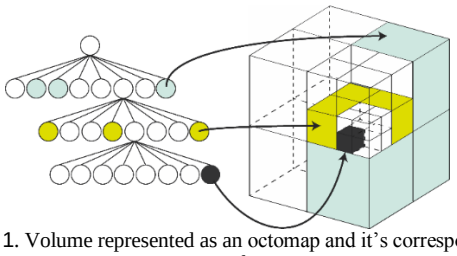

Figure 1. Volume represented as an octomap and it's corresponding octree scheme

Generic quadrotor UAVs in our simulation perform 3D mapping of imported landscapes with Kinect RGBD-sensors in order to demonstrate a success of map import with our tool. Quadrotor simulation is based on ROS-package *hector_quadrotor* [10]. Kinect sensors are located at quadrotor bottom with a 45-degree pitch angle (Fig.2, left). This angle helps avoiding occlusions (with propellers) and is optimal for object detection under a flying quadrotor.

Husky robot (ROS-package *husky_robot)* checks collision detection while moving through synthetic landscape and is fully operational in ROS framework [11]. As a second platform for map verification we employ Turtlebot robot (ROS-package *TurtleBot*, is shown in Fig. 2, right) – a simple moving base with sensors that has libraries for visualization, planning, perception and control, which is featured with a simpler moving and collision physics compared to Husky.

C. Sensor-based input data

In order to create a realistic simulation and verify the proposed solution we used a real data environment map. The data was obtained with a mounted on top of a Pioneer ground mobile robot laser range finder while navigating through Autonomous Navigation and Perception Lab (ANPL) facility at the Technion - Israel Institute of Technology (Fig.3, left). The second map (Fig.3, right) was obtained from Hackaday website (https://hackaday.com/tag/lidar/). The maps are built using the *gmapping* code (ROS package) and can be loaded by using *rosservice* from *map_server* ROS package. The resulting maps present an occupancy grid that could be visualized as a grayscale image in PGM format. It has three possible pixel values of "black" for occupied space, "white" for free space and "gray" for vaguely defined space that means uncertain situation with no scanning data available. In order to support path planning, maps should be filtered before importing it to Gazebo environment.

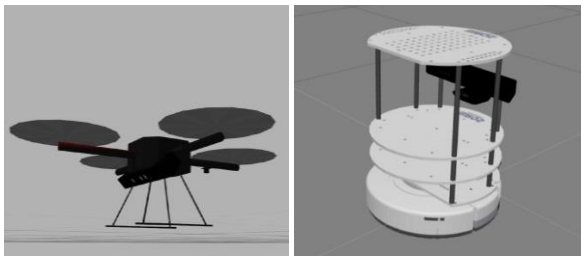

Figure 2. Generic quadrotor model with a Kinect sensor (on the left), Turtlebot robot model in Gazebo (on the right side)

III. MAP FILTERING

Various existing filters for image noise reduction could be roughly classified into linear, non-linear and fuzzy-classical filters [12]; particularly interesting for our task is a simple salt-and-pepper filter [13]. In this section we briefly overview these filter types and justify the selection of a best suitable for our task filter. Throughout the section we keep a uniform notation for an image *A* that consists of pixels *(i,j)* with *(i,j)*'s grey-value denoted by *A(i,j)* and a corresponding filtered image *A'*.

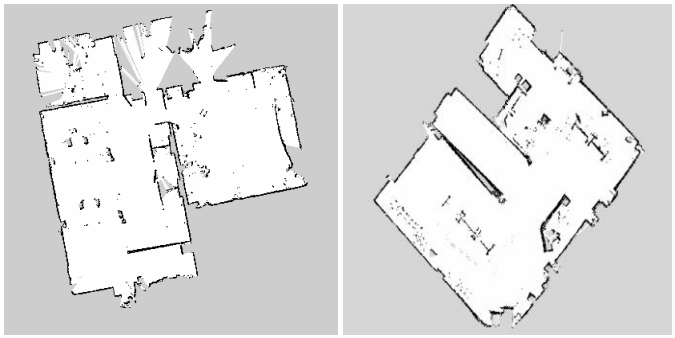

Figure 3. Original sensor-based maps

A. Linear filters

Linear filters filter images by replacing an original pixel value with a linear combination of the pixel's neighborhood. Linear filters are easy to implement but all of them blur an image, making obstacle edges smoother. A new pixel value is

calculated as a weighted average of the values within a square region centered at a pixel (i,j):

$$A'(i,\ j) = \sum_{k=-N}^{N} \sum_{l=-N}^{N} w(k,l) \cdot A(i-k,\ j-l) \quad (1)$$

where N is a size of a square mask of the filter and weight coefficients w(k,l) satisfy the condition:

$$\sum_{k=-N}^{N} \sum_{l=-N}^{N} w(k,l) = 1 \qquad (2)\ .$$

B. Non-linear filters

Non-linear filters have several improvements to cope with disadvantages of linear filters. Adaptive mean filters are designed to detect if pixels belong to a same homogeneous region. To do that, the algorithm counts the difference between a proceeded pixel of interest and its neighbor. If they belong to a same region, their mutual influence is significant, otherwise – very low. This dependency is expressed with the equation:

$$w_{ij}(k,l) \sim |A(i,j) - A(i-k, j-l)| \qquad (3)$$

Such filters better preserve object edges but do not completely eliminate a blurring effect.

Median filters work in slightly different way, applying pixel neighborhood median value; they are ideal for reducing extreme pixel values and successfully deal with impulse noises:

$$A'(i,j) = \text{median}_{-N \le k,l \le N}\, A(i-k, j-l) \qquad (4)$$

Main drawback of median filters is that thin edges and lines may be entirely eliminated during filtering process.

C. Fuzzy-classical filters

Fuzzy-classical filters apply particular mathematical functions (e.g., mean, median or a more sophisticated function) and holistic rules to calculate weights of each neighbor pixel, considering difference between pixels' locations and values. Commonly, large value difference or distance decrease pixel weight and vice versa.

Fuzzy-classical filters may be the most agile way to reduce noises on a wide range of images. However, optimal function selection and implementation of these filters may be rather cumbersome.

D. Salt-and-pepper filter

Salt-and-pepper filters are used mainly to reconstruct image after data loss during a delivery or file damaging [13]. Algorithms of these filters look for "extreme" values, which are called "salt" and "pepper" if pixel values are abnormally low or high respectively. Filtering consists of two main stages: detection stage and filtering stage.

Detection stage searches noisy pixels - local minima ("salt") and maxima ("pepper") of intensities starting from boundary pixels. The second stage is actual filtering when algorithm starts to gradually increase neighborhood area in order to make a filtered value as much accurate as possible. Increasing stops when it becomes impossible to do that without new noisy pixels inclusion. This constraint minimizes influence of noisy pixels on each other.

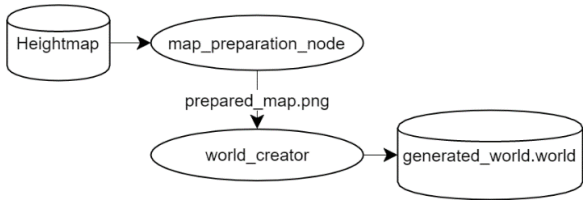

Figure 4. Filtering ROS node data flow scheme

E. Filter selection

Original sensor-based mapping marks pixels as "white" (free space), "black" (obstacle) or "grey" (unknown pixel that had been invisible for the sensor). Thus, noisy pixels contain exactly the same values variety options as pixels with real values. Moreover, all noises are impulse noises, i.e. they are not uniformly distributed within a map.

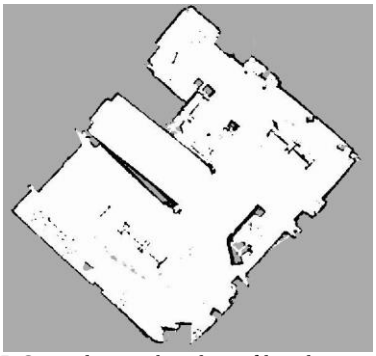

Figure 5. Original sensor-based map filtered using median filter

Having only three types of pixel values makes impossible to use all filters, which produce mean filtered values, as they cannot be interpreted in terms of pixel occupancy. Fuzzy-classical filters may perform slightly better, but require a significant amount of time to adapt to particular map properties. Salt-and-pepper filters may seem to be the best in reducing impulse noises but their weakness is noisy pixel detection stage, as it is impossible to classify noise by its value because there is no difference between noisy and noise-free pixels' values.

Therefore, we use a non-linear median filter, which does not produce meaningless values, effectively reduces impulse noises and is easy to implement. The filter was implemented as ROS node written in C++ language. It receives an occupancy grid as an input, filters it and outputs an occupancy grid in a same format (Fig. 4 represents the algorithm flow). All phases - original map processing, filtering and filtered map saving - were combined into a single launch-file, which makes filtering and further import easy. The filtered map is demonstrated in Fig. 5.

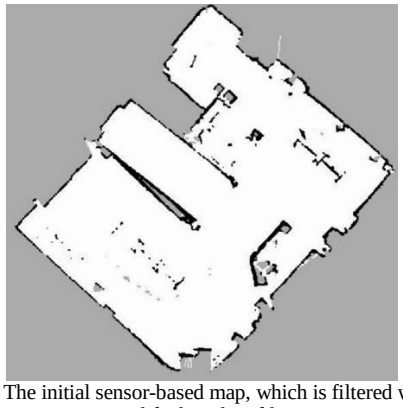

Figure 6. The initial sensor-based map, which is filtered with the use of modified median filter

However, we noticed that thin structures, such as inner walls, were eliminated during the filtering process. To cope with this, we modified our median filter implementation not to affect pixels *(i,j)* which closest elements in row have the same value as pixel of interest itself. We denoted them as *(i,j-1), (i,j+1) (i-1,j)* in row case and as *(i+1,j)* in column case. The result of filtered map is demonstrated in Fig. 6.

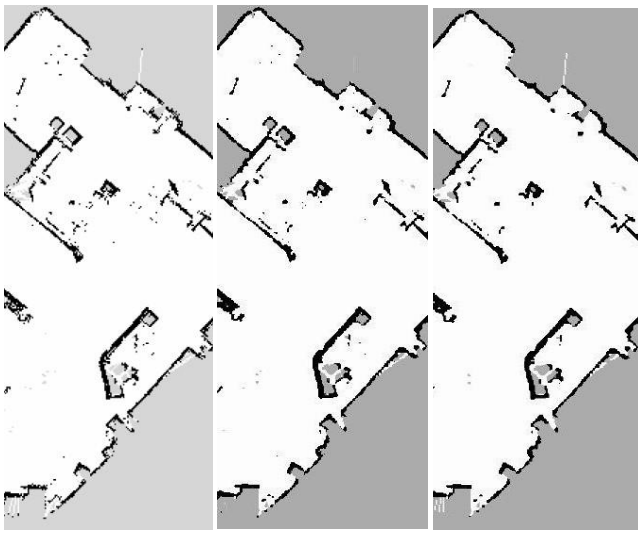

Figure 7. Scaled region of the initial map (left) is filtered with use of median filter (on the middle), and modified median filter (on the right).

Modified filter effectively preserves thin structures in heightmap. Difference between initial map and two filtering results is demonstrated on Fig. 7

IV. MAP IMPORTING

To store object descriptions in a scene, parameters of physics, collision detection, etc., Gazebo uses special world SDF format file [14]. Initially, within our project such SDF world was populated with manually created objects (traffic cones, walls etc.), which were stored as 3D-models with textures (Fig. 8). However, this solution could not be applied directly for automatic import of a sensor-based map into SDF world. To import such map as a landscape we verified several options, which are described in details in this section.

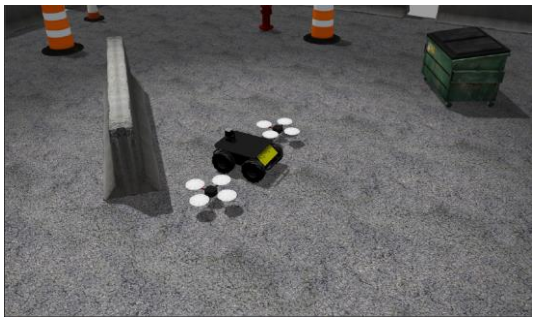

Figure 8. Initial project with Husky and two quadrotors

A. Automatic map to 3D-model conversion

Automatic map to 3D-model conversion method was tested using third-party command line tool called *stl_tools* [15]. This tool uses PNG image as an input, processes it and outputs STL model. Figure 9 demonstrates a resulting model of original sensor-based map (Fig.3, left). However, this model cannot store textures and measuring units, as well as does not interact with light or shades in the simulator.

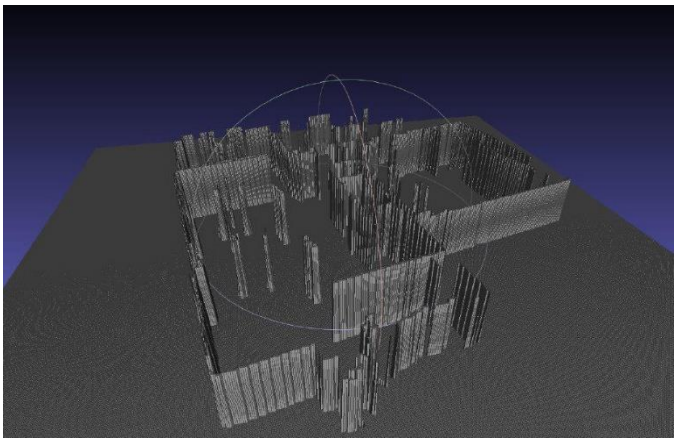

Figure 9. Resulting STL model from *stl_tools*

B. Automatic world file generation

Automatic world file generation is implemented as a ROS node, which takes a filtered occupancy grid and creates a world that is populated with box-type objects that correspond to occupied areas coordinates.

However, when the node was implemented and launched, we discovered an undocumented constraint of Gazebo simulator that does not allow to have more than one thousand distinct objects in a scene simultaneously. Thus, as the filtered original map contains over 3000 occupied grid cells, the automatic generator succeeded to support only a partial map. While it may

be possible to optimize the script by merging distinct neighboring occupied cells, such solution is not generally scalable. In addition, there is no direct way to explicitly mark uncertain areas ("grey" pixels).

C. Heightmap construction

We build an octomap type map using *SDF*-element *heightmap*, which requires base image file for environment construction and allows specifying landscape size and its maximum height within its XML format. The attempt to utilize *heightmap* directly failed to create a valid landscape as in addition to issues with obstacle heights (the automatic conversion inverted all heights) it does not support robot collision treatment (Fig. 10 shows the resulting invalid map). This was caused by requirements of *heightmap* on base image file to be in grayscale mode PNG format instead of a standard default RGB mode.

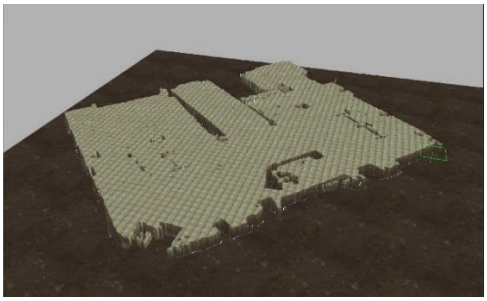

Figure 10. RGB image importing result: inverted heights

Figure 11. Successfully imported heightmap with textures

Our tool fixes this issue by performing file conversion of a preprocessed with a filtering map from RGB to PNG format prior to employing a *heightmap.* As an output the tool provides a valid map that supports proper textures, colors and heights for occupied, free and unknown areas as well as collision, light and shading treatment. Figure 11 demonstrates the resulting final map of the filtered original map (Fig.3, right).

D. Map verification stage

To verify the usability and quality of the created with our tool landscape, we tested it for a navigation task with the mentioned above Husky robot and two quadrotors. While testing, we measured performance of Gazebo simulation with Real Time Factor (RTF), which shows a ratio of execution time and calculation time within a simulation for a particular task. For example, if it takes 4 seconds to compute 1 second of execution in the simulation, RTF equals 0.25. Naturally, the larger is RTF, the more efficient is the simulation.

Several optional robot models were used for verification. For the two quadrotors performing a navigation task within the simulated landscape the RTF was equal to 0,7. For Husky robot together with the two quadrotors the RTF decreased to unacceptable rate of 0,01, while for Tutrlebot robot [16] with the two quadrotors the RTF decreased to acceptable value of 0,3. This confirmed our assumption about a particularly complicated interaction between the resulting heightmap and Husky robot model. Figure 12 demonstrates a simulation of navigation task for a Tutrlebot robot and two quadrotors within a resulting environment that was obtained from the filtered map (Fig. 5).

Figure 12. A final map with two quadrotors and Tutrlebot, performing a joint navigation task

V. TOOL PARAMETERS

Multiple parameters allow to define cropping offsets; minimal desired image height and width; filtering and color inversion options. The implemented in tool options are:

map_filepath – path for processing the initial heightmap.

saving_path – path for processing the map saving.

use_filtering – filters image with modified median filter.

color_inverse – replaces every pixel with its additional color (Fig. 13).

cropped_width / *cropped_height* – crops (or extends) the image as defined by the passed sizes. In case of extension, additional color area changes the color mode to the grayscale.

offset_x and *offset_y* – crop the area of vertical and horizontal image respectively, that are defined by passed value. The example of changing offsets is shown on Figure 14.

Only *map_filepath* option is necessary, all others are optional and implemented for easy usage.

After launching, the initial heightmap becomes cropped (or extended) and changes the color mode to the grayscale. Next steps, which include filtering and color inverse, are optional. For further use the image need to be saved in PNG format. Finally, Gazebo file is created and can be used by Gazebo simulator to launch. For easy usage all tool features are integrated with ROS.

Tool launching example with multiple options:

roslaunch gazebo_heightmap_preparation start.launch map_filtering:=home/project/ANPL_map.pgm

saving_filepath:= home/project use_filtering:=true cropped_width:=400 color_inverse:=false offset_x:=1830 offset_y:=1800

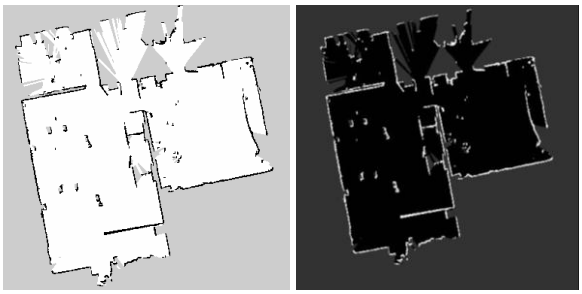

Figure 13. Tool launching with multiple options

Figure 14. Tool launching with multiple options

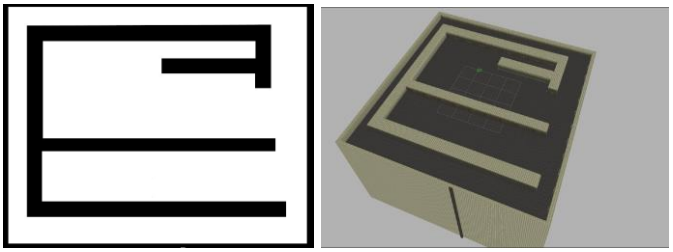

Figure 15. Example of tool execution – from initial image (left) to Gazebo environment (right)

VI. CONCLUSIONS

In this paper, we presented the development of an automatic tool that allows creating realistic landscapes in Gazebo simulation from arbitrary images (Fig. 15) and from a real world sensor-based exploration map. The tool provides automatic filtering and import of an occupancy grid map into Gazebo framework as a heightmap. As a part of our future work, we plan to verify the obtained tool for a set of existing in ROS models of robots as well as for the ROS-model of Engineer robot [6]. The tool will be further improved and compiled as an open source ROS-package for public domain.

REFERENCES

[1] Jorge Fuentes-Pacheco, José Ruiz-Ascencio, Juan Manuel Rendón-Mancha. Visual simultaneous localization and mapping: a survey. Artificial Intelligence Review, November 2015.

[2] A. Buyval, I. Afanasyev, and E. Magid, Comparative analysis of ROS-based Monocular SLAM methods for indoor navigation, The 9th Int. Conf. on Machine Vision, 2016.

[3] R. W. Wolcott, R. M. Eustice. Visual Localization within LIDAR Maps for Automated Urban Driving, 2014.

[4] I. Afanasyev, A. Sagitov, and E. Magid. ROS-Based SLAM for a Gazebo-Simulated Mobile Robot in Image-Based 3D Model of Indoor Environment. Advanced Concepts for Intelligent Vision Systems, Springer International Publishing, pp.273-283, 2015.

[5] V. Indelman, L. Carlone and F. Dellaert, ""Planning in the continuous domain: A generalized belief space approach for autonomous navigation in unknown environments.", The Int. Journal of Robotics Research, vol. 34, no. 7, pp. 849-882, 2015.

[6] M. Sokolov, R. Lavrenov, A. Gabdullin, I. Afanasyev and E. Magid. 3D modelling and simulation of a crawler robot in ROS/Gazebo. The 4th Int. Conf. on Control, Mechatronics and Automation, 2016.

[7] Open Source Robotics Foundation, "nav_msgs/OccupancyGrid Documentation," http://docs.ros.org/kinetic/api/nav_msgs/html/msg/OccupancyGrid.html

[8] S. Singh, R. Simmons, T. Smith, A. Stentz, V. Verma, A. Yahja & K. Schwehr. Recent Progress in Local and Global Traversability for Planetary Rovers. IEEE Int. Conf. on Robotics and Automation, 2000.

[9] Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., & Burgard, W. (2013). OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, *34*(3), 189-206.

[10] S. K. Johannes Meyer, "hector_quadrotor - ROS Wiki," http://wiki.ros.org/hector_quadrotor.

[11] Clearpath Robotics, "Robots/Husky - ROS Wiki," http://wiki.ros.org/Robots/Husky

[12] Nachtegael, M., Van der Weken, D., Van De Ville, D., Kerre, E., Philips, W., & Lemahieu, I. (2001). An overview of classical and fuzzy-classical filters for noise reduction. *The 10th IEEE Int. Conf. on Fuzzy Systems* (Vol. 1, pp. 3-6), *2001*.

[13] Toh, Kenny Kal Vin, and Nor Ashidi Mat Isa. "Noise adaptive fuzzy switching median filter for salt-and-pepper noise reduction." *IEEE signal processing letters* 17.3 (2010): 281-284.

[14] Open Source Robotics Foundation, "SDF," http://sdformat.org/

[15] T. Hearn, "GitHub - thearn/stl_tools: Python code to produce STL geometry files from plain text, LaTeX code, and 2D numerical arrays (matrices)," https://github.com/thearn/stl_tools

[16] M. W. Tully Foote, "Robots/TurtleBot - ROS Wiki," http://wiki.ros.org/Robots/TurtleBot

[17] E. Magid, T. Tsubouchi. Static Balance for Rescue Robot Navigation : Discretizing Rotational Motion within Random Step Environment. Lecture Notes In Artificial Intelligence, Vol. 6472, Proceedings of International Conference on Simulation, Modeling, and Programming for Autonomous Robot, pp. 423-435, 2010.

[18] E. Magid, T. Tsubouchi, E. Koyanagi, T. Yoshida. Building a Search Tree for a Pilot System of a Rescue Search Robot in a Discretized Random Step Environment, Journal of Robotics and Mechatronics, Vol.23(1), August 2011, pp.567-581.