Automatic Mapping and Filtering Tool: From A Sensor-Based Occupancy Grid to a 3D Gazebo Octomap

Roman Lavrenov, Aufar Zakiev, Evgeni Magid
Intelligent Robotics Dept., Higher Institute for Information Technology and Information Systems (ITIS)
Kazan Federal University
Kazan, Russian Federation
e-mail: lavrenov@it.kfu.ru, zaufar@gmail.com, magid@it.kfu.ru

***Abstract*—Robot simulations nowadays provide significant support in testing new algorithms for robotic systems for a broad area of tasks, including navigation, mapping, and SLAM. Within a simulation, special attention should be paid for providing algorithms with realistic testing environments that are to be further used for robot navigation. This paper presents an automatic tool that allows creating realistic landscapes in Gazebo simulation, which are based on results of real world sensor-based exploration. The tool provides automatic filtering and importing of an occupancy grid map into Gazebo framework as a heightmap.**

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

I. INTRODUCTION

Mobile robotics is an important branch of robotics field, which have demonstrated a significant and continuous growth over past decades. One of the basic difficulties that mobile robotics researchers and developers daily face is navigation problem. Navigation could be roughly divided into three main areas: localization, mapping and path planning [1]. Localization is responsible for exact determining of robot's current position in an environment. Mapping consists of collecting sensory data and storing it in a form that is convenient for further processing. Path planning searches a route within an environment from start to goal position while utilizing data that was collected during a mapping process. These three tasks are highly dependent; for example, a proper localization provides a more precise object positioning within a map, while a good map increases localization accuracy. In order to resolve these dependencies simultaneous localization and mapping (SLAM) techniques were developed. SLAM unites localization and mapping processes to make both more effective and agile. Existing 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] and others [1].

Navigation process and, in particular, SLAM methods should be carefully tested before integrating them into real 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]. One of the most critical requirements for a simulated environment is its high realism, i.e. both simulated landscapes and objects should provide a good approximation to real world scenarios. The best way to create such simulated environment is to employ real sensor-based input data. For our research project on localization, mapping and path planning for a UGV with an aid of a UAVs group using active collaborative vision and multi-robot belief space planning, we are utilizing Robot Operating System (ROS) framework. However, ROS 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. Finally, we conclude in Section 5.

II. MOTIVATION AND SYSTEM SETUP

This section describes our long-term research goal that had motivated the development of the presented in this paper simple and convenient tool for real sensor-based data import and processing. Next, we introduce setup of the simulation and sensor-based input data description.

A. Project Motivation

Our long-term research goal 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 latter could be completely unknown, include dynamic scene or objects, or being represented by an outdated and 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 investigate such a multi-robot collaborative framework with the aim to advance the state of the art in multi-robot autonomy under uncertainty with vision and additional sensors such that the robots could autonomously operate under partial and possibly uncertain information regarding the environment. We build upon recent progress in SLAM and belief space planning [5], and will explore multi-robot belief space planning approaches that take into account the uncertainty in the environment. In particular, such a framework will allow UAVs and the UGV to devise proper motion to improve localization and mapping even in lack of GPS signal.

978-1-5090-6530-1/17/$31.00 ©2017 IEEE

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. In order to create a realistic synthetic environment by translating real sensory data into a ROS map for further path planning, we have developed an easy-to-use tool for ROS and Gazebo, presented in this paper, which may become useful for all ROS users that feel the need for various sensor-based simulated environments in their research.

B. Simulation Setup

Robot Operating System (ROS) framework serves as an effective tool, which among broad variety of tasks supports mapping and data visualization. For our research, we use ROS Indigo 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).

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.

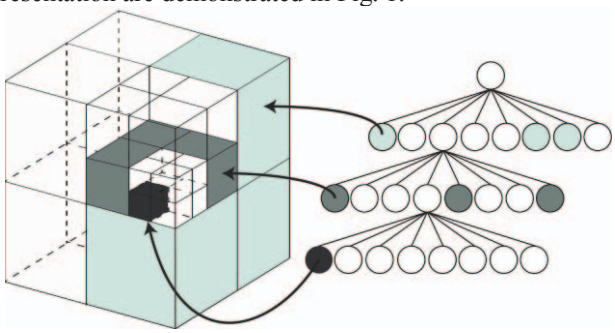

Figure 1. Volume represented as an octomap and it's corresponding octree scheme

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*, ROS-model is shown in Fig. 2, right) 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*) – 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.

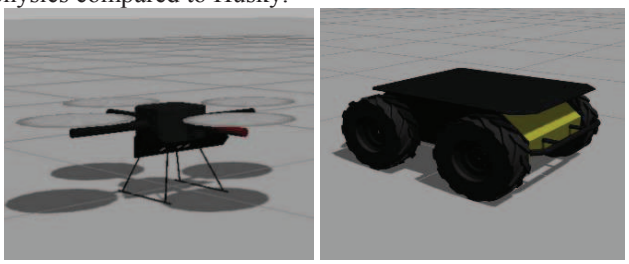

Figure 2. Generic quadrotor model with a Kinect sensor (on the left), Husky robot model in Gazebo (on the right side)

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. The map is built using the *gmapping* code (ROS package) and can be loaded by using *rosservice* from *map_server* ROS package. The resulting map presents an occupancy grid that could be visualized as a grayscale image in PGM format (Fig. 3). 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 when no scanning data available about this region.

Figure 3. Original sensor-based map

The advantage of such map is availability of extracting each particular pixel information with ROS tools, while its main disadvantage is a presence of noise that is caused by the nature of real sensor-based data (encapsulated with red rectangulars in Fig. 4). Thus, in order to support path planning, a map should be filtered before importing it to Gazebo environment.

Figure 4. Scaled region of the original map with noisy regions (several examples are encapsulated within red rectangulars)

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 is denoted by A'.

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:

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

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 following equation:

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

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:

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

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 (Fig. 5).

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.

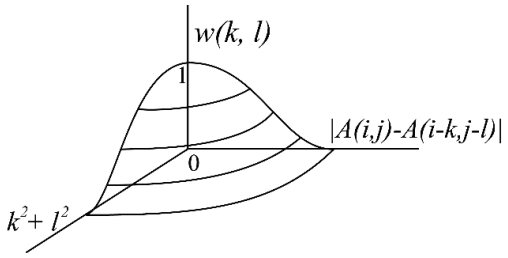

Figure 5. The weight function in fuzzy filters

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.

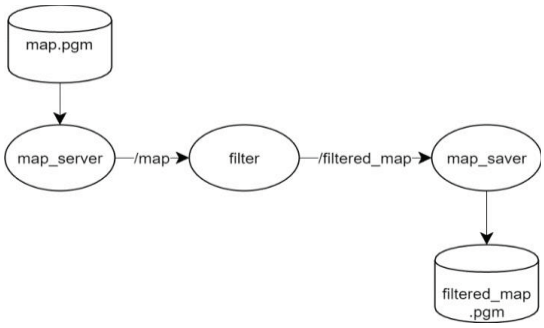

Figure 6. Filtering ROS node data flow scheme

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.

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. 6 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 resulting filtered map is demonstrated in Fig. 7.

Figure 7. Filtered map of the original sensor-based map

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.

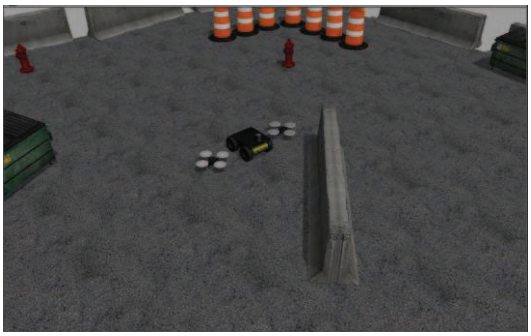

Figure 8. Initial project screenshot

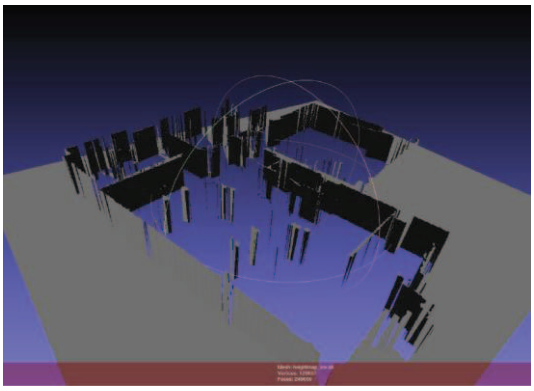

Figure 9. Resulting STL model of the original sensor-based map

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). However, this model cannot store textures and measuring units, as well as does not interact with light or shades in the simulator.

B. Automatic World File Generation

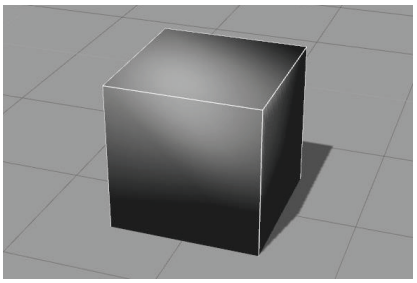

Figure 10. Typical "box" object in Gazebo

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 (Fig. 10).

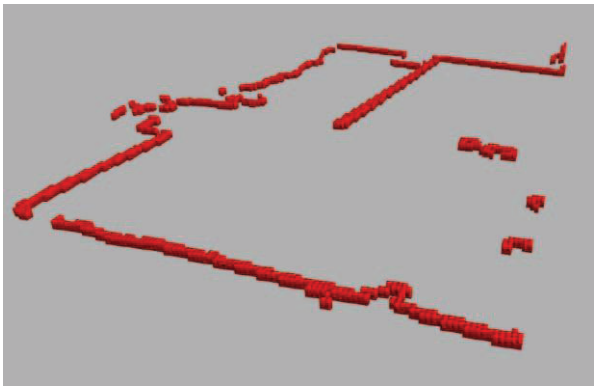

Figure 11. An automatic world generation n- a part of the original sensor-based map, that was simulated with 1000 box objects

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 (Fig. 11). 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. 12 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.

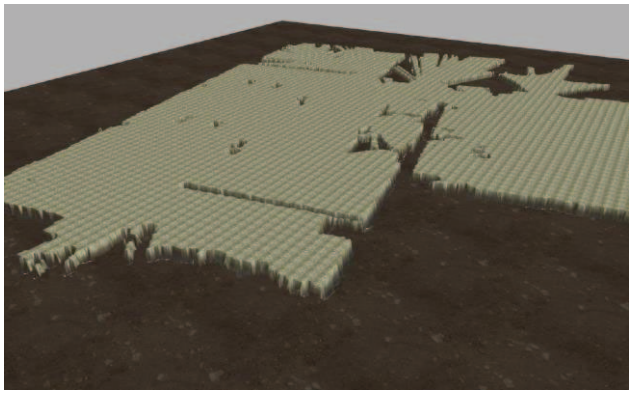

Figure 12. RGB image importing result: inverted heights

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 13 demonstrates the resulting final map of the filtered original map (Fig. 3).

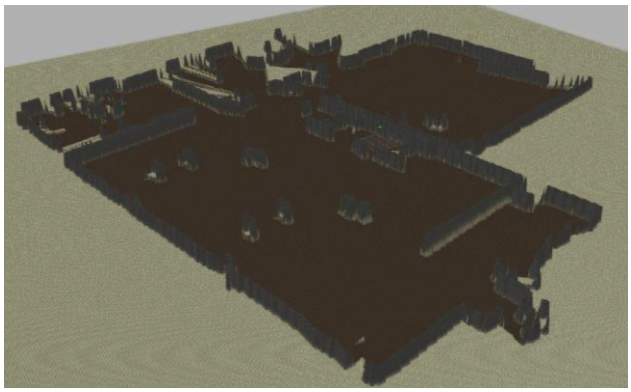

Figure 13. Successfully imported heightmap with textures

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.

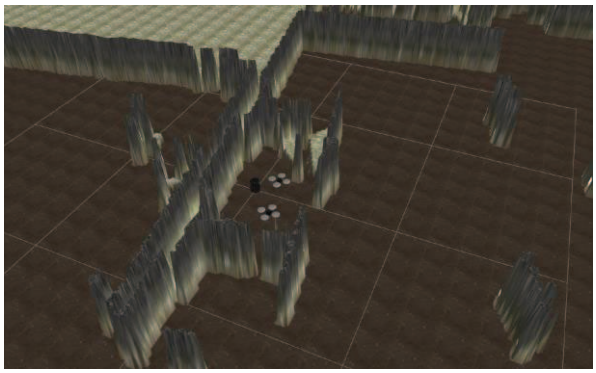

Figure 14. A final map with two quadrotors and Tutrlebot, performing a joint navigation task

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 14 demonstrates a simulation of navigation task for a Tutrlebot robot and two quadrotors within a resulting environment that was obtained from the sensor-based map (Fig. 7).

V. CONCLUSIONS

In this paper, we presented the development of an automatic tool that allows creating realistic landscapes in Gazebo simulation 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.

ACKNOWLEDGMENT

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.

REFERENCES

[1] J. Fuentes-Pacheco, J. Ruiz-Ascencio, and J.M. Rendón-Mancha, Visual simultaneous localization and mapping: a survey. *Artificial Intelligence Review*, *43*(1), pp. 55-81.

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

[3] R.W. Wolcott, R.M. Eustice, Visual Localization within LIDAR Maps for Automated Urban Driving, *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2014),* 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, 2015, pp. 273-283.

[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 International Journal of Robotics Research*, vol. 34, no. 7, 2015, pp. 849-882.

[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 International conference on Control, Mechatronics and Automation (ICCMA)*, Barcelona, Spain, 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, and K. Schwehr, Recent Progress in Local and Global Traversability for Planetary Rovers. *Proceedings of the IEEE International Conference on Robotics and Automation,* 2000.

[9] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, W. Burgard, OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, *34*(3), 2023, pp. 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] M. Nachtegael, D. Van der Weken, D. Van De Ville, E. Kerre, W. Philips, and I. Lemahieu, An overview of classical and fuzzy-classical filters for noise reduction. *The 10th IEEE International Conference on Fuzzy Systems,* Vol.1, 2001, pp. 3-6.

[13] K.K.V. Toh, and N.A.M. Isa. Noise adaptive fuzzy switching median filter for salt-and-pepper noise reduction. *IEEE signal processing letters*, 2010, pp. 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