Network failure detection and autonomous return algorithms for a crawler mobile robot navigation

Neil Alishev
Intelligent Robotic Systems Laboratory
Kazan Federal University
Kazan, Russia
alishev.neil@gmail.com

Roman Lavrenov
Intelligent Robotic Systems Laboratory
Kazan Federal University
Kazan, Russia
lavrenov@it.kfu.ru

Kuo-Hsien Hsia
Department of Electrical Engineering
Far East University
Tainan City, Taiwan
khhsia@mail.feu.edu.tw

Kuo-Lan Su
Department of Electrical Engineering
National Yunlin University of Science and Technology
Douliu City,Taiwan
sukl@yuntech.edu.tw

Evgeni Magid
Intelligent Robotic Systems Laboratory
Kazan Federal University
Kazan, Russia
magid@it.kfu.ru

***Abstract*—In this paper we present an algorithm for a mobile robot autonomous return. The algorithm involves a network failure detection module, which is based on analysis of incoming UDP packets. Simultaneous Localization and Mapping (SLAM) and path planning algorithms were used as an integral part of the autonomous return algorithm. The algorithms were integrated into Russian mobile robot Servosila Engineer, and experiments were conducted in order to determine the best configuration of the algorithm parameters.**

Keywords—Ground mobile robot, algorithm, autonomous return, network failure detection, path planning, ROS.

I. INTRODUCTION

Robots often operate in conditions that are hardly suitable for human beings, including blockages, contaminated areas, and narrow spaces. Some robots use tether communication between a robot and a human operator, while other robots rely on wireless communication. In a case of the network failure or tether abruption there exists a high risk of losing a robot. In the course of robot exploitation network failure may occur in the following cases: thick walls or other obstacles between a robot and an operator; strong magnetic or radio interference; long distance between a robot and an operator so that the robot goes out of a range of a radio transmitter.

In this research, we developed and integrated into Russian mobile robot Servosila Engineer control system our autonomous return algorithm, which helps solving the problem of robot loss in a case of network failure. Problem of autonomous return was decomposed into two subproblems [1-3] (Fig. 1): Simultaneous Localization and Mapping (SLAM) on the outward way and autonomous navigation on the return way. On the outward way, when network connection functions properly, the robot is being teleoperated to a target point. During that procedure, the robot activates SLAM algorithm, stores a generated map and localizes itself on that map. When our algorithm detects that network connection is lost, the robot uses path-planning algorithm to autonomously navigate back to the starting point.

Efficiency and accuracy of an autonomous return algorithm depends on the efficiency and accuracy of underlying algorithms. For example, if SLAM algorithm is not providing detailed maps, the robot will fail to accurately reach the starting point. In addition, SLAM algorithm should be resource efficient in order to run locally on the robot. Same holds true for all other algorithms, including the network failure detection algorithm. There are multiple ways to detect network connection disruptions between robot and the operator. In this research we present a way to detect the network failure, which is both accurate and resource efficient in our robot system setup. Also, this algorithm doesn't use any additional devices and rely only on the existing robot hardware.

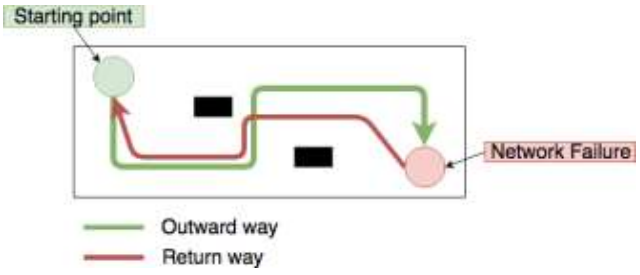

Figure 1. Autonomous return illustration.

978-1-5386-6712-5/18/$31.00 ©2018 IEEE
DOI 10.1109/DeSE.2018.00040

II. SYSTEM SETUP

Servosila Engineer (Fig. 2) is a Russian crawler-type mobile robot [4], which was designed for various applications including urban search and rescue operations, operations within dangerous or inaccessible for a human environments and other areas. The robot in its original configuration is equipped with an optical zoom camera and a pair of stereo vision cameras. The robot is resistant to harsh conditions and is water and dust proof. It is operated with the original interface in a teleoperation mode only and an operator controls velocities of servos and positions of the robot's parts.

Figure 2. Servosila Engineer crawler-type robot with a Hokuyo UTM-30LX-EW LRF mounted on its top.

For SLAM purposes we used Hokuyo UTM-30LX-EW laser range finder (LRF). We designed and constructed with a 3D printer a special static stand for the LRF with an option to select an inclination of a scanning beam toward the locomotion surface of an environment. Due to mounting with adjustable angle it is possible to use the LRF in overcoming obstacles and solving the problems of static and dynamic balance [5]. The stand was attached to the top of the robot head (Fig. 3).

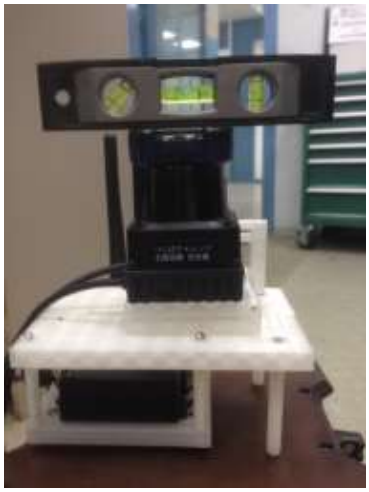

Figure 3. Hokuyo laser on a stand. Laser is parallel to the surface of the floor.

In addition to the original server and software, which were provided by the manufacturer, we installed Robot Operating System (ROS), Indigo version, in order to run ROS nodes, which are required for the LRF output data streaming and LRF SLAM algorithms. Even though newer versions of ROS, e.g., Kinetic Kame, are already available, we were restricted to use ROS Indigo version because Sevosila Engineer robot operates with Ubuntu 14.04, which is specially tailored to the robot's operator interface, hardware, and drivers.

III. NETWORK FAILURE DETECTION ALGORITHM

Before implementing our own network failure detection algorithm, we researched existing approaches to solving this problem. In [6-7] researchers utilized a special device for measuring a signal strength. When the signal strength drops below some threshold it is considered unacceptable and a special behavior is triggered (e.g., stopping robot's servo drives). Figure 4 shows a relation between the signal strength and robot's distance from a radio transmitter. Three distance zones with regard to the signal strength are distinguished: a safety zone, a controlled zone and a forbidden zone. The safety zone is an area where the signal strength is acceptable and a robot functions properly. The controlled zone is an area where the special behavior is triggered. The forbidden zone is an area with a low level of signal, which disappears completely as the distance increases.

Derbakova et.al. in [8] solved a problem of network failure detection when a single human operator controls multiple robots simultaneously. In their configuration robots were connected to the operator and also were connected to each other. Instead of using a special signal strength measuring device the researchers monitored a hop count value of packets that were passing through the robots of the group. When one of the robots experienced network problems and went down the overall hop count value decreases by 1.

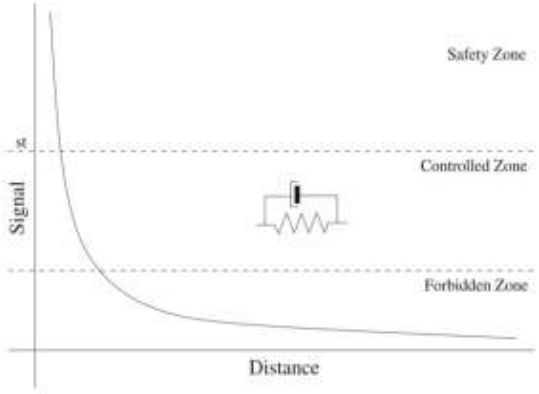

Figure 4. Illustration that shows relation between signal strength and distance from the radio transmitter [6].

Hsieh et.al. [9] were sending units of data every time interval. They defined a successful transaction as a transmission of such a unit of data by a sender followed by a receipt of acknowledgment, which was further sent by a receiver. Then, based on a desired transaction rate, i.e., number of successful transactions per time interval, the sender, periodically evaluates its connection with the receiver.

Our approach to the network failure detection is coming from our system setup and Servosila Engineer robot's software features. To implement network failure detection algorithm, we started off by investigating Servosila remote control protocol that is responsible for communication between a user and the robot during the teleoperation. It turned out that this protocol consists of two independent system processes:

- Vehicle process
- Operator Control Unit (OCU) process

The vehicle process runs on an on-board control computer of the robot. It receives commands from an OCU and executes the commands by commanding servo drives and chassis motors of the robot. The vehicle process also sends a compressed video stream and telemetry information back to the OCU computer.

The OCU process runs on a portable OCU computer. It transmits joystick commands to the vehicle process running on an on-board control computer of a robot. The OCU process receives and displays via a graphical user interface a video stream and telemetry information coming from the vehicle process. The vehicle and OCU processes communicate with each other by exchanging UDP packets. The diagram in Fig. 5 shows a remote control protocol in action.

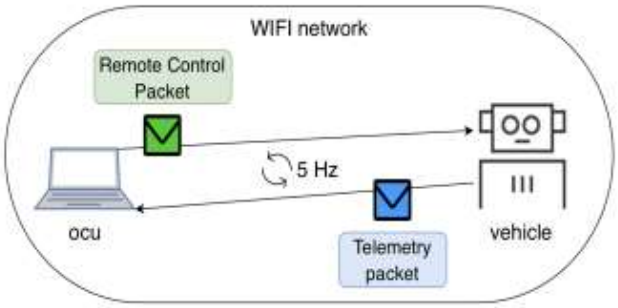

Figure 5. Servosila Remote Control Protocol diagram.

Remote Control Packet is a packet that an operator's device is sending to the robot's computer. This packet contains all information regarding robot movement (e.g., values of desired robot axis). We use a self-made Graphical User Interface to convert human inputs into the axis angle values and send it as a remote control packet [19]. Telemetry packet, on the other hand, is a packet that the robot sends back to the operator, and it contains data from the robot's encoders.

If the vehicle process does not receive any packet from an OCU within an interval of 1.00 second, the vehicle process assumes that a radio communication to the OCU have been lost. The vehicle process then automatically shuts down all servo motors and chassis motors for safety. This means that the OCU process has to continuously send packets to the vehicle process in order to maintain continuous motion of the robot's motors. In our GUI, which converts user inputs to remote control packets, we send packets to the vehicle process with the 5 Hz rate. This rate was selected empirically during experiments. It turned out that at remote control packet sending rates lower than 5 Hz the robot cannot be operated smoothly. It is also worth noting that those packets are sent even if there are no input commands from the user.

Considering our robot system setup and remote control software features we decided to monitor incoming UDP packets to detect a network failure. The task was to detect incoming UDP packets, which are remote control packets, and to monitor them in time. If for some time (e.g., 5 seconds) there are no incoming remote control packets, it means that the connection with the client is lost and the robot should switch to autonomous return mode. To parse incoming UDP packets we created a python script (Fig. 6), which gets an incoming packet and determines if it is a UDP packet. Next, a UDP packet is verified for being a remote control packet. When a connection is restored (e.g., when the robot approaches the operator in autonomous return mode), incoming UDP remote control packets are detected and a manual control is returned to the operator. The *move base* function goal of returning to the starting point is been canceled.

Our network failure detection algorithm does not require an additional device in order to detect signal strength and does not consume significant system resources. This is possible due to the fact that we do not send additional "is alive" packets to the robot, but use existing remote control packets that are already being sent.

```
# packet string from tuple
packet = packet[0]

# parse ethernet header
eth_length = 14
eth_header = packet[:eth_length]
eth = unpack('!6s6sH', eth_header)
eth_protocol = socket.ntohs(eth[2])

protocol = None

# UDP packets
if protocol == 17:
    u = iph_length + eth_length
    udph_length = 8
    udp_header = packet[u:u + 8]
    # unpack header
    udph = unpack('!HHHH', udp_header)
    source_port = udph[0]
    dest_port = udph[1]
    length = udph[2]
    checksum = udph[3]
    if dest_port == 10000:
        rospy.loginfo("Remote control packet received")
```

Figure 6. Remote control UDP packet parsing code snippet.

IV. AUTONOMOUS RETURN

A network failure detection algorithm is only a part of the autonomous return algorithm. To successfully return to a starting point after a network failure, we need to perform SLAM-based navigation on the outward way and to perform path planning and autonomous navigation on the return way. The workflow of our system as whole is shown in Fig. 7.

First step in establishing SLAM for the robot was to get a data stream from the LRF. ROS package *urg_node* was used for these purposes. This package allows to run a ROS node, which reads data from the LRF and publishes LaserScan messages to the */scan* topic. Next, *laser_scan_matcher* package [10] was used to obtain odometry from laser scans. This

package compares consecutive *LaserScan* messages to estimate position of the LRF in space. Laser-based odometry turned out to be quite accurate. After getting odometry, we aimed to perform mapping and localization for our robot.

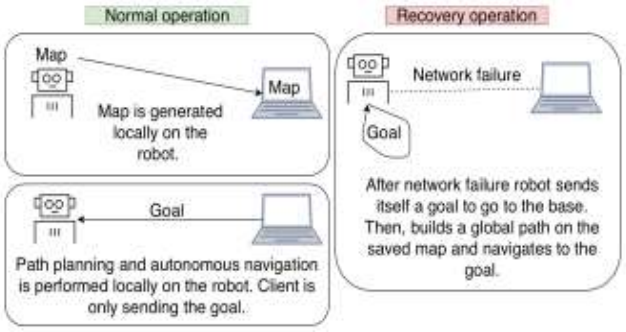

Figure 7. Our system workflow diagram.

Before running SLAM algorithms [11] on the real robot, we conducted a set of trials in Gazebo simulation in order to identify the most suitable for our purposes LRF-SLAM algorithm [12]. We had tested *gmapping* [13], *Google Cartographer* [14] and *Hector SLAM* [15] algorithms on prerecorded rosbagfiles with laser range data and concluded that *Gmapping* and *Hector SLAM* significantly outperformed *Google Cartographer* algorithm. Thus, *gmapping* and *Hector SLAM* algorithms were selected as the LRF-SLAM algorithms for further experiments with the real robot.

Gmapping SLAM package is implemented using Rao-Blackwellized particle filter. Each particle in this algorithm is a separate Dynamic Bayesian Network that stores its' own version of a map. Rao-Blackwellized particle filter is applied to these particles in order to pick out the most plausible information about the environment [16]. Also, the process called marginalization is used to reduce the number of particles. This process solves the main challenge when using the particle filter. It groups similar particles into one particle in some area R. This reduces the number of particles and allows faster execution of the algorithm and lower memory costs [17]. *Hector SLAM*, on the other hand, uses a different approach to perform mapping and localization of a robot. A 2D robot pose is estimated based on a scan matching process. The scan matching algorithm used in *Hector SLAM* is based on Gauss-Newton approach. The algorithm seeks to find the optimum alignment of laser scan's endpoints with the constructed map by finding a rigid transformation $\xi = (px,py,\psi)^T$ that minimizes:

$$\sum_{i=1}^{n}[1 - M(S_i(\xi))]^2$$

where the function $M(S_i(\xi))$ returns the map value at $S_i(\xi)$, which is the world coordinates of the scan endpoint.

After establishing *gmapping* for our robot with default parameters, real-world experiments were conducted in order to determine the quality of mapping and localization. During the experiments, the robot was teleoperated from a start point to a target point while performing LRF-SLAM and a generated map was stored. First, RBPF–SLAM algorithm was tested. RBPF–SLAM algorithm has a lot of parameters, which depend on laser frequency and range, odometry information source and level of sensor shaking when the robot moves. These parameters should be finely tuned for each particular case in order to obtain the best mapping and localization quality. Figure 8 demonstrates a map of a same area (doorway), that was obtained in two experiments: before and after parameter tuning. The left picture shows a map defect in the area behind a left wall, while the picture on the right does not contain such defect. Parameter estimation and RBPF–SLAM algorithm configuration was performed during numerous experiments in different environments.

Next, *Hector SLAM* experiments were conducted. After several experiments, although RBPF-SLAM parameters were finely tuned, it turned out that in our setup *Hector SLAM* algorithm gives better results than the RBPF–SLAM. Thus, all subsequent experiments were conducted using *Hector SLAM* algorithm. The long corridor was selected as a next experiment location because such locations are the most challenging ones when an odometry is coming from a LRF. In the experiment, the robot was teleoperated along the corridor from the start point to the target point, and the generated map was stored. Although the corridor itself does not contain a large number of feature points, algorithm succeeded to construct a map without any significant errors. After that, *Hector SLAM* algorithm was used to create a map of the entire building floor. The resulting map is demonstrated in Fig. 9.

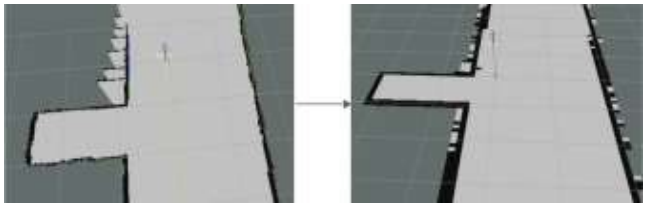

Figure 8. Experiments within the same area before and after parameters tuning.

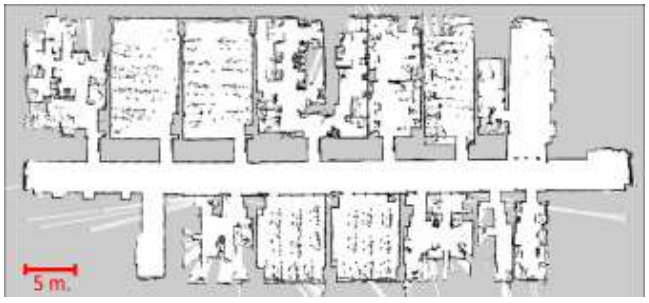

Figure 9. Higher Institute for Information Technology and Information Systems floor map. Created in a single run. Legs of school desks are marked as multiple points in periodic shapes.

After establishing SLAM for our robot on the outward way we proceeded to path planning and autonomous navigation. For path planning we used A* algorithm, which demonstrated the best performance during experiments with regard to other path planning algorithms. It was used to build a path to the starting point when a connection with the operator was lost. The path was built on the map, which had been generated on the outward way. The example of path is shown in Fig. 10.

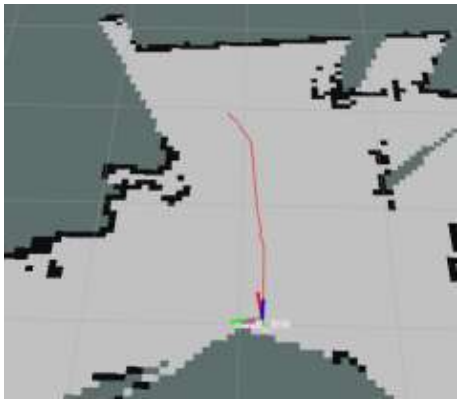

Figure 10. A* algorithm path example.

In ROS, the best practice for implementing autonomous navigation is implementing the *move_base* [18] module. To implement this module, we completed the following steps:

- *local_costmap* and *global_costmap* configurations are used to correctly track obstacles in the environment. In addition, robot footprint information falls into this configuration.
- *local_planner* and *global_planner* configurations ase used to build local and global path plans.
- Encapsulate previous configurations into *move_base* module.

When we publish a goal to the *move_base* module, it starts to publish messages to the *cmd_vel* topic. This topic contains velocity in free space being split into its linear and angular parts. After completing above-mentioned steps, we got *move_base* module output as *geometry_msgs/Twist* type message in the *cmd_vel* topic.

The next step in building autonomous navigation for the robot is implementing *base_controller*. This module is subscribing to the *cmd_vel* topic and converts Twist velocity messages to individual commands for the robot's servo drives. This step is the most challenging because *base_controller* implementation is different from one robot to another and depends greatly on a robot's physical arrangement. The fact that Servosila Engineer is a crawler-type robot made the task more complicated since we cannot map directly between *cmd_vel* velocity commands (e.g., turning left with the speed of 0.1 m/s) and robot's servo drive commands. Therefore, all conversions between *cmd_vel* velocities and real servo drive commands were done through numerous experiments.

After completing *base_controller* we aimed to the real-life navigation experiments. We set *move_base* goal and monitored robot's autonomous navigation accuracy. At first, the robot was lacking accuracy, sometimes accidentally went into recovery behavior mode and sometimes oscillated back and forth in the final point because fault tolerance values were not set. Experimentally we picked up configuration values, which gave us the best navigation accuracy. The configuration values are shown in Table 1.

TABLE I. NAVIGATION CONFIGURATION VALUES

Parameter name	Parameter value
max_vel_x	0.6 m/s
min_vel_x	0.2 m/s
max_vel_y	0.0 m/s
min_vel_y	0.0 m/s
min_rot_vel	0.2 rad/s
acc_lim_theta	10.0 rad/s^2
acc_lim_x	10.0 m/s^2
sim_time	0.5 s
yaw_goal_tolearnce	0.15 rad
xy_goal_tolearnce	0.15 m
path_distance_bias	50
trans_stopped_vel	0.2 m/s
rot_stopped_vel	0.2 rad/s
stop_time_buffer	0.1 s
dwa	true
meter_scoring	true
holonomic_robot	false

V. CONCLUSION

Autonomous return is an important feature that helps to prevent losing a mobile robot due to disruption of its wired or wireless connection with the operator. In this research we proposed the algorithm of autonomous return, which involves network failure detection algorithm, SLAM algorithm, navigation and path planning algorithms. Combination of these algorithms allowed us to solve the task of autonomous robot return in a case of a network failure. All algorithms were integrated into the Russian mobile robot Servosila Engineer control system and are launched automatically on robot startup.

ACKNOWLEDGMENT

This work was partially supported by the Investment and Venture Fund of the Republic of Tatarstan, project ID 13/43/2018. This work was performed according to the Russian Government Program of Competitive Growth of Kazan Federal University.

REFERENCES

[1] Shimizu, T., Awai, M., Yamashita, A., & Kaneko, T. Mobile robot system realizing human following and autonomous returning using laser range finder and camera. Proceedings of the 18th FCV, pp. 97-102, 2012.

[2] Tsuda, N., Harimoto, S., Saitoh, T., & Konishi, R. Mobile robot with following and returning mode. In Robot and Human Interactive Communication. The 18th IEEE International Symposium on, pp. 933-938, 2009.

[3] Awai, M., Shimizu, T., Kaneko, T., Yamashita, A., & Asama, H. Hog-based person following and autonomous returning using generated map by mobile robot equipped with camera and laser range finder. In Intelligent Autonomous Systems 12, pp. 51-60, 2013.

[4] Sokolov, M., Afanasyev, I., Lavrenov, R., Sagitov, A., Sabirova, L., & Magid, E. (2017). Modelling a crawler-type UGV for urban search and

rescue in Gazebo environment. In Artificial Life and Robotics (ICAROB 2017), International Conference on, pp. 360-362, 2017.

[5] Magid E. & Tsubouchi T. Static balance for rescue robot navigation: Discretizing rotational motion within random step environment. In Int. Conf. on Simulation, Modeling, and Programming for Autonomous Robots, pp. 423-435, 2010.

[6] Tardioli, D., Mosteo, A. R., Riazuelo, L., Villarroel, J. L., & Montano, L. Enforcing network connectivity in robot team missions. The International Journal of Robotics Research, 29(4), pp. 460-480, 2010.

[7] Mong-ying, A. H., Cowley, A., Kumar, V., & Taylor, C. J. Towards the deployment of a mobile robot network with end-to-end performance guarantees. In Robotics and Automation, ICRA 2006. Proceedings 2006 IEEE International Conference on, pp. 2085-2090, 2006.

[8] Derbakova, A., Correll, N., & Rus, D. Decentralized self-repair to maintain connectivity and coverage in networked multi-robot systems. In Robotics and Automation, 2011 IEEE International Conference on, pp. 3863-3868, 2011.

[9] Hsieh, M. A., Cowley, A., Kumar, V., & Taylor, C. J. Maintaining network connectivity and performance in robot teams. Journal of field robotics, 25(1-2), pp. 111-131, 2008.

[10] Laser scan matcher, ROS package: http://wiki.ros.org/laser_scan_matcher

[11] Ohno, K., & Tadokoro, S. Dense 3D map building based on LRF data and color image fusion. In Intelligent Robots and Systems (IROS 2005), pp. 2792-2797, 2005.

[12] Alishev, N., Lavrenov, R., & Gerasimov, Y. Russian mobile robot Servosila Engineer: designing an optimal integration of an extra laser range finder for SLAM purposes. In Int. Conf. on Artificial Life and Robotics, pp. 204-207, 2018.

[13] Quigley, M., Stavens, D., Coates, A., & Thrun, S. Sub-meter indoor localization in unmodified environments with inexpensive sensors. In Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on, pp. 2039-2046, 2010.

[14] Barnard, K. D. (2002). U.S. Patent No. 6,456,938. Washington, DC: U.S. Patent and Trademark Office.

[15] Kohlbrecher, S., Meyer, J., Petresen, K., & Graber, T. Hector SLAM for robust mapping in USAR environments. ROS RoboCup Rescue Summer School Graz, 2012.

[16] He, M., Takeuchi, E., Ninomiya, Y., & Kato, S. Precise and efficient model-based vehicle tracking method using Rao-Blackwellized and scaling series particle filters. In Intelligent Robots and Systems, 2016 IEEE/RSJ International Conference on, pp. 117-124, 2016.

[17] Doucet, A., De Freitas, N., Murphy, K., & Russell, S. Rao-Blackwellised particle filtering for dynamic Bayesian networks. In Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence, pp. 176-183, 2000.

[18] move_base, ROS package: http://wiki.ros.org/move_base

[19] Mavrin, I., Lavrenov, R., Svinin, M., Sorokin, S., & Magid, E. Remote control library and GUI development for Russian crawler robot Servosila Engineer. In MATEC Web of Conferences (Vol. 161, p. 03016), 2018.