Development of a Graphical User Interface for a Crawler Mobile Robot Servosila Engineer

Mavrin Ilya
Intelligent Robotic Systems Laboratory
Kazan Federal University
Kazan, Russia
i.a.mavrin@gmail.com

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

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

***Abstract*—Nowadays some manufacturers in an attempt to decrease a product price and this way to increase its competitiveness on the global market offer robots with high-quality hardware, but with a very basic software. For this reason, a user often cannot exploit all available functionality of a robot. These circumstances force users to develop their own software, aimed to add some new features and fix bugs of the original software. In this paper we present graphical user interface development for Russian crawler robot Servosila Engineer.**

Keywords—GUI, mobile robot, crawler robot, 3D model, position control, software development.

I. Introduction

Nowadays as robots receive a wide spread and gradually turn from an expensive unique "piece of art" into a mass-product. For this reason, some manufacturers in an attempt to decrease its price and this way to increase its competitiveness on the global market offer robots with high-quality hardware, but do not make a sufficient effort to develop its software. As a result, a user often cannot exploit all available functionality of a robot. However, such drawbacks of manufacturer`s original software should not force every user to purchase a new expensive robot as an experienced user could re-implement some parts of original software and to add new functionalities in order to maximize the hardware usage.

The main complexity that an experienced user, which takes a challenge of updating and upgrading robot original software on his/her own risk, immediately faces is the fact that typically manufacturer`s software is not an open-source code and the manufacturer only provides limited documentation of their application programming interface (API). Thus, a user has to invent nontrivial solutions to comply his/her own software with manufacturers API. Moreover, while it takes significant amount of time to implement such walk-around solutions, some internal features will be still unreachable for a user.

In our case, we successfully completed the task of implementing a new graphical user interface (GUI) for Servosila Engineer crawler robot [14] on a client side in order to send out remote control packets from user`s PC to the robot via Wi-Fi. Yet, re-implementing a server side is a much more complicated task, because its software should interact with motors and sensors directly. Here we also faced a challenge of implementing an accurate joint position controller for the robot, which could accept an absolute angle value for each joint and perform the corresponding actions of servos.

The rest of the paper is organized as follows. Section II introduces the original system setup, including both hardware and software components. Section III overviews a number of existing GUIs of mobile robots and manipulators. Section IV presents our own GUI development, including velocity and position control functionality, 3D model and video streaming widgets. Finally, we conclude in Section V.

II. Original system setup

A. Robot hardware

Engineer is a crawler-type mobile robot, manufactured by Russian company Servosila for operating in difficult conditions (Fig. 1). The robot has a waterproof and dustproof body frame and radiation proof electronics. Servosila Engineer robot is tooled with a 5 degree of freedom manipulator with a gripper, which can grasp objects, open doors and perform various actions with objects. The robot is featured with a rich pack of cameras and sensors, including optical zoom camera, a rear camera and stereo vision cameras. Most of the sensors and CPU are packed into the robot head, which is located on the end effector of the manipulator. Unfortunately, it is impossible to get data from an onboard IMU using the original API. In addition to the original sensors, we added Hokuyo UTM-30LX-EW laser range finder [15] to allow laser-based Simultaneous Localization and Mapping (SLAM) and path planning algorithms integration into robot upper-level control system.

B. Original software

The original software is built as a client-server architecture. The server is an application, which starts running on an embedded computer within the robot head immediately after switching on the robot. This server controls motors, receives information from the built-in IMU, sends video from cameras and a telemetry of some of the servos to a client via Wi-Fi connection. A client is an application, which starts on a user's PC and sends commands to the server. A user can control the

978-1-5386-6712-5/18/$31.00 ©2018 IEEE
DOI 10.1109/DeSE.2018.00044

robot using Xbox 360 gamepad, which should be calibrated before each use.

Fig. 1. Servosila`s Engineer robot (courtesy of the manufacturer)

TABLE I. REMOTE CONTROL PACKET

Field	Size	Type
Frame type ID	1 byte	uint8
Axis #0	2 bytes	int16
Axis #1	2 bytes	int16
…	…	…
Axis #15	2 bytes	int16
Button #0	1 byte	uint8
Button #1	1 byte	uint8
…	…	…
Button #15	1 byte	uint8
Video bit rate telemetry	8 bytes	double
Total size	57 bytes	

The original GUI consists of two widgets. The first one is a video streaming widget, which contains only one feature of camera selection for video streaming. The second widget is the robot 3D model demonstration, which has no customizations and contains a number of obvious drawbacks. For example, a user cannot rotate the 3D model and in some cases the model may fall apart after a number of transformations. This software uses a simple control protocol that consists of two fixed size Telemetry (Table 1) and Remote control packets (Table 2) and varying size Video Frame packet (Table 4). The telemetry packet contains Motor data structures (Table 3) for each motor.

TABLE II. TELEMETRY PACKET

Field	Size	Type
Frame type ID	1 byte	uint8
Tick number	8 bytes	int64
Number of motors	1 byte	uint8
Motor data #0	24 bytes	struct
Motor data #1	24 bytes	struct
…	…	…
Motor data #9	24 bytes	struct
Not used	25 bytes	-
Total size	275 bytes	

TABLE III. MOTOR DATA STRUCTURE

Field	Size	Type
Device ID	1 byte	uint8
Device state	1 byte	uint8
Operation mode	1 byte	uint8
Position	4 bytes	uint32
Speed	2 bytes	int16
Electric current (in amperes)	2 bytes	int16
Status bits	2 bytes	int16
Position command	4 bytes	uint32
Speed command	2 bytes	int16
Electric current command (in amperes)	2 bytes	int16
Total size	24 bytes	

TABLE IV. VIDEO FRAME PACKET

Field	Size	Type
Frame type ID	1 byte	uint8
JPEG image	varying size	-
Total size	depends on frame size	

Remote control packets (RCP) are strongly tied to buttons and joystick axes of Xbox 360 gamepad. In general, the original RCP for Servosila Engineer robot just represents a state of a gamepad. It is a critical limitation, because it is possible to

perform only those commands, which are available through the Xbox gamepad. RCPs are sent five times per second by the client. The server receives RCPs and sends back Telemetry packets to the client. Some packets can get lost during a transfer, because they are sent via UDP (User Datagram Protocol). If the server doesn`t receive RCPs more than 1 second, the robot stops. The telemetry packets are aimed to return information about the robot motors. These packets contain motor data structures, one structure for each motor. The structure contains data about a motor, such as its state, position, velocity, electric current etc. The most useful information is a position of a motor shaft. Having this information is enough to implement a simple position controller, which compares current and desired positions, and activates the motor if needed. Thus, the most critical drawbacks of the original control protocol include the following issues:

- The control protocol strongly depends on Xbox360 gamepad. Therefore, any GUI, which uses the protocol could perform only those commands, which are allowed by the original GUI using this gamepad, and this way any GUI that is based on the original protocol has to emulate an interaction with the gamepad.
- Position control can be implemented only on a client side, because the protocol allows to send only velocity values to the motors, which decrease the accuracy of the position controller.
- Inability of checking the battery and the headlight status. In the case of the headlight a user can visually (i.e., through one of the cameras) check if the light is switched on/off. In the case of the battery a user should avoid long lasting remote tasks, because the robot may suddenly switch off as the result of the battery depletion. Moreover, if the robot switches off due to the battery depletion inside a narrow sub-human void (e.g., a pipeline or a tunnel [16]), it will be almost impossible to bring it back or extract it with a help of another robot. Moreover, to check the battery level a user has to connect an additional display to the embedded computer of the robot.
- Inability of customizing video quality in wide ranges (e.g., a video resolution, FPS, a video codec). The original protocol allows to change only the bitrate.
- Inability to rotate the gripper using the original protocol.
- The original protocol does not allow to rotate a waist joint and a chassis simultaneously. So the robot cannot rotate the waist while being in motion and has to stop at first. We assume that it is a side effect of a primitive restriction-based self-collision avoidance system of the server.

These drawbacks prevent to implement all features that should provide a comfortable level of a teleoperation process. Thus the original protocol should be rewritten and the server should be re-implemented.

III. Overview of Graphical User Interfaces

Since our robot combines a mobile crawler chassis with a manipulator, in this section we overview a number of existing GUIs of mobile robots as well as static manipulators in order to analyze the trends and to select potentially interesting solutions that could be integrated into the new GUI of Servosila Engineer robot. Due to the lack of space we had to constraint ourselves to present only a limited number of the existing solutions.

Figure 2 demonstrates a GUI for controlling a mobile robot with a 3 degrees of freedom (DoF) manipulator, which was implemented in Matlab environment [1]. It provides basic functionality for moving the mobile chassis in four directions, controlling all joints and the gripper, and turning on/off the torch. Its video features include video streaming, recording, and saving video data. A useful feature of this GUI is the possibility to explicitly reset each joint as well as the chassis.

Fig. 2. A GUI a mobile robot with a 3 DoF manipulator [1]

Fig. 3. A GUI for a mobile robot with an obstacle plot [2]

Figure 3 presents a GUI for a mobile robot [2] that has a number interesting features, including an obstacle plot, a video streaming widget and “stop all” button. We decided to integrate “stop all” into our new GUI for Servosila Engineer robot (a spacebar hotkey) and to add obstacle plot feature as a “to do” task for the next version of our GUI.

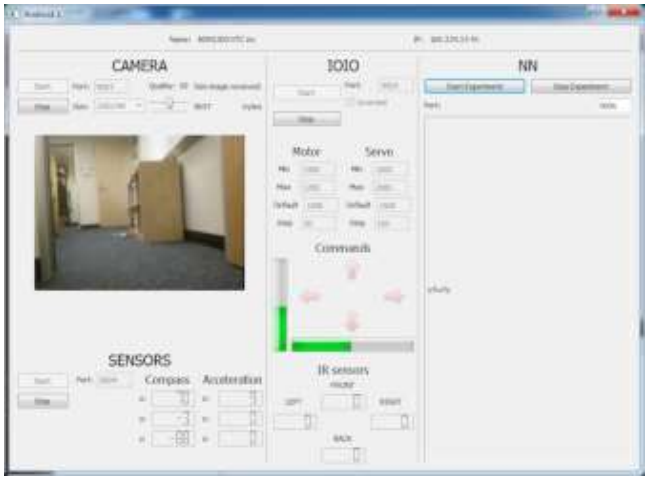

Fig. 4. GUI for Android based robot [3]

Figure 4 presents a GUI of an Android based mobile robot [3] that contains a video streaming widget, sensory data and motor controllers. It enables selection of a video stream quality, control of minimal, maximal and default motor velocities as well as a step size.

Fig. 5. GUI of Helpmate robot [4]

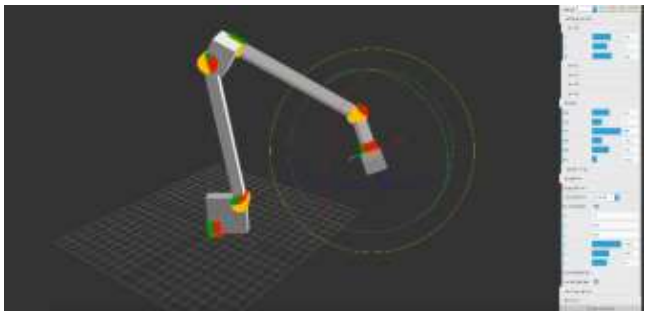

Fig. 6. A "Robot-gui" project [5]

Figure 5 presents a GUI of Helpmate mobile robot [4], which has a sonar and a lidar widgets, a video streaming widget and a colored obstacle map. The GUI has a multiple window interface, but we consider this uncomfortable for a practical use, because an operator typically utilizes a number of widgets simultaneously, and moving and scaling these windows in an attempt to arrange them without overlapping should be avoided.

Figure 6 presents a GUI of static 6-joint manipulator [5], which animates a 3D structure of the robot using OpenGL [6]. This solution inspired us to implement a 3D widget for our robot. Figure 7 presents a GUI of a differential drive robotic rover [7], which has a number of useful features, including a 3D model widget, a radar widget and a 2D arm control widget.

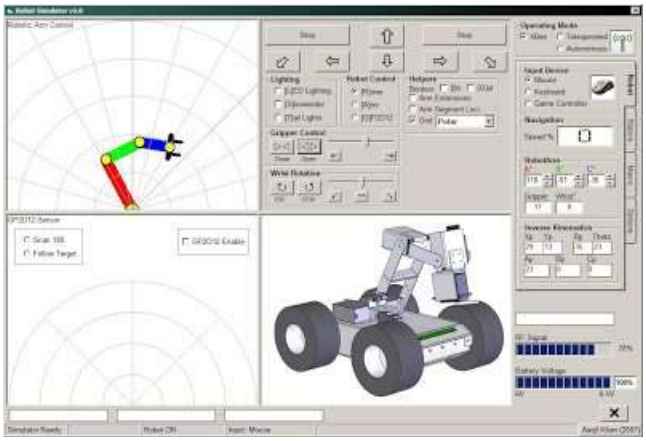

Fig. 7. GUI of a differential drive robotic rover [7]

After GUI review we have decided to add the following modules and functionality to the first version of our own GUI for Servosila Engineer robot [8]:

- A 3D model widget
- A widget for video from cameras
- An obstacle map
- A sonar data widget
- A lidar data widget

While the first widget was recently integrated into the GUI and the second widget development is our on-going work, the other three widgets are left as a part of our future work.

IV. Our GUI development

This section describes the components of our GUI, including a velocity controller (which sets motors velocities), a position controller (which sets absolute angles of the motors) and a 3D model widget (which shows the current state of the robot). We also describe an on-going development of a video streaming widget, which can show video from all cameras of the robot.

A. Velocity control

A velocity control module (Fig. 8) sends velocity values for each robot motor via a Wi-Fi connection [8]. The velocity controller has different interfaces: to control the robot a user can use hotkeys, sliders or can print velocity values directly into text boxes. Also user can define velocities, which will be sending to robot by pressing a hotkey.

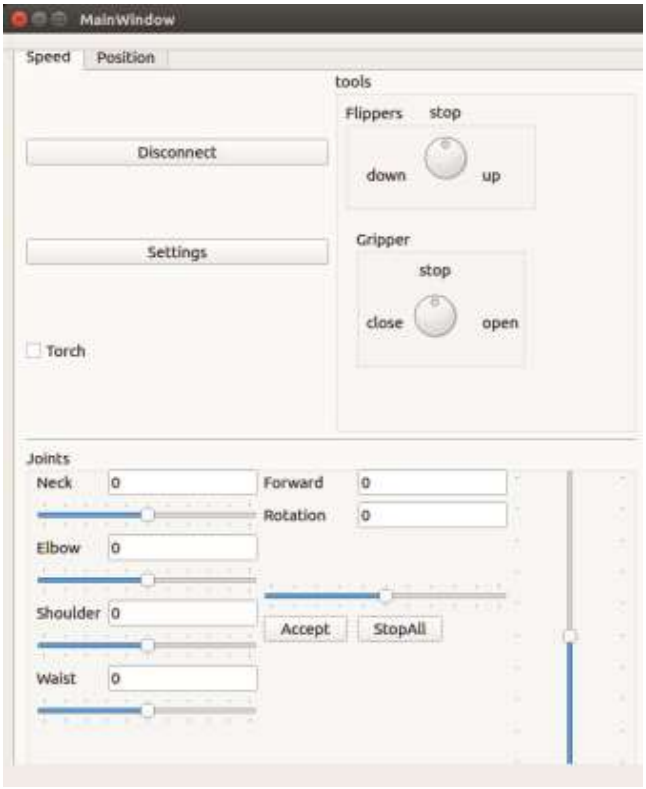

Fig. 8. Velocity control module

B. Position control

A position control module (Fig. 9) allows to control each joint using its absolute angle value. It can be extremely useful in integration with ROS. For example, it allows to use inverse kinematics. Because of some disadvantages of Servosila`s original API it is impossible to send position values directly to the robot, and GUI is able to send only velocity values. Because of this reason, position is computed on a user`s PC using telemetry from the robot and a desired angle.

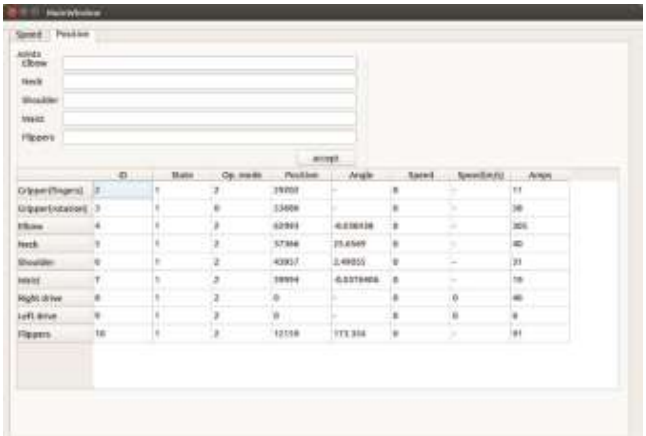

Fig. 9. Position control module

C. 3D model widget

A manufacturer kindly shared a high polygonal 3D model of Engineer robot (Fig. 10) with our team, which was further used for creating a GUI widget that demonstrates a position of each joint (Fig. 11). To implement a render engine, we used Assimp library [9] for 3D model loading and OpenGL for rendering. As a skeletal animation will be redundant for simple transformations, we splitted the robot 3D model into meshes (each moveable part of the robot formed a standalone mesh) and rotated these meshes. Centers of each mesh were set to their rotation axis, on a joint connection point. Mesh splitting was performed in Blender [10]. The engine required a child-parent tree structure, because some moveable parts are located on other moveable parts. Like in forward kinematics, child meshes should move together with parent meshes. Assimp library provides the child-parent tree with a relative (to the parent) transform matrices for each mesh on load. So the engine should traverse the child-parent tree and calculate the absolute transform matrices. The mesh (as an array of vertices) and its absolute and relative transformation matrices are stored together in Mesh class. A function, which implements rotation of a joint, simply rotates the desired mesh around its local coordinates. Child joints are rotated for the same angle around the desired mesh local coordinates. After a rotation the function recomputes the absolute transform matrices. At the same time relative transform matrices are always constant. Mesh transformation is implemented as applying rotation to each vertex of the mesh in a cycle. The cycle is parallelized with OpenMP [12] library for better performance.

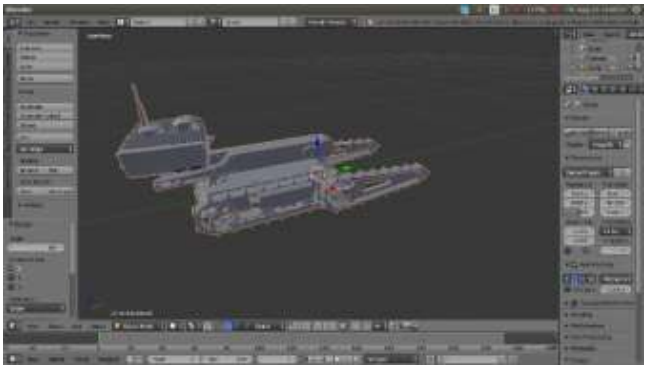

Fig. 10. 3D model from Servosila

D. Video streaming widget

A video streaming widget in the original software has only an ability to select a single camera among the four cameras of the robot for data streaming. Moreover, Fig. 12 demonstrates that the video has a greenish color palette and suffers from a fisheye effect [17]. Since such video data does not allow a comfort teleoperation, we decided to develop new widget that could show video from all four cameras simultaneously or from a single selected camera, but in a better quality. For this task we need to develop our own video server, because using the original protocol we can obtain a video stream only from a single camera. Our video server should have frame preprocessing to deal with greenish color palette and fisheye effect, and to allow a flexible

adjustment of video quality (FPS, resolution, bitrate, video codec etc.), because a bandwidth of radio channel is usually not enough to stream video from all four cameras in best available quality simultaneously, which is our on-going work [11].

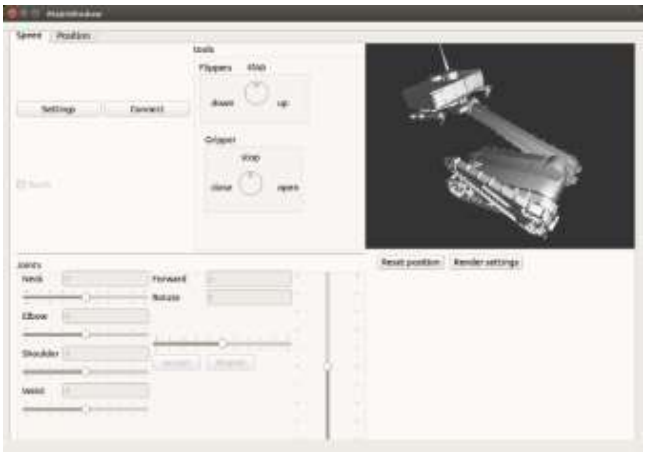

Fig. 11. Our GUI with the 3D model widget

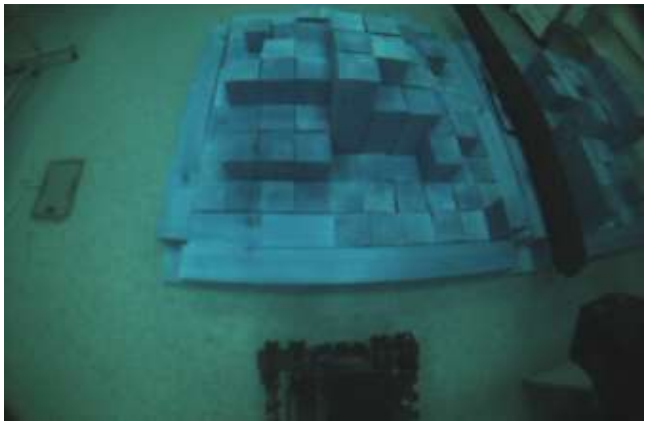

Fig. 12. A video from a camera with the original software

V. Conclusions and Future Work

If an original software package that automatically comes with a newly purchased robot lacks some important features, it is not always necessary to purchase a new robot as experienced users could take a challenge and attempt developing their own software in order to add new features and fix bugs of the original software. In this paper we presented a graphical user interface development for Russian crawler robot Servosila Engineer, that had been caused by our desire to improve the original graphical user interface and to ease the teleoperation process of the robot. In particular, we replaced original joystick-only control with a GUI that contains multiple control options for motor velocities, position control and 3D widget that demonstrates the current state of the robot in real-time.

Our on-going efforts deal with implementation of video streaming widget that will allow to transfer and view video data from all four cameras simultaneously in a real-time. Since our future projects concentrate on collaboration and path-planning tasks [13] for Servosila Engineer robot, we are planning to develop obstacle map widget, which will be updated in real-time, and a pilot advisory system that will help to keep the robot balance in order to avoid rollover [18].

Acknowledgment

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

References

[1] Chhaniyara K. "GUI for controlling robot" https://www.mathworks.com/matlabcentral/fileexchange/24239-gui-for-controlling-robot/

[2] Kerry M., "A layared approach to designing robot software", in National Instruments, 2012.

[3] Oros N., Krichmar JL., "Smartphone based robotics: Powerful, flexible and inexpensive robots for hobbyists, educators, students and researchers", in IEEE Robotics & Automation Magazine, 2013.

[4] http://eecs.vanderbilt.edu/CIS/IRL/helpmate.shtml/

[5] Beck M., project "robot-gui" https://github.com/glumb/robot-gui/

[6] OpenGL project, https://www.opengl.org/

[7] Khan A., blog http://aaqilkhan.blogspot.ru/2007/08/

[8] 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, EDP Sciences, 2018.

[9] Assimp library, http://www.assimp.org/

[10] Blender 3D creation suite, https://www.blender.org/

[11] Safin, R., Lavrenov, R., Saha, S. K., & Magid, E. Experiments on mobile robot stereo vision system calibration under hardware imperfection. In MATEC Web of Conferences, Vol. 161, p. 03020, EDP Sciences., 2018.

[12] OpenMP library, http://www.openmp.org/

[13] Panov A., Yakovlev K. Behavior and path planning for the coalition of cognitive robots in smart relocation tasks. Robot Intelligence Technology and Applications 4, pp. 3-20., Springer, Cham, 2017.

[14] 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.

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

[16] Murphy, R. R. Human-robot interaction in rescue robotics. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 34(2), pp. 138-153, 2004.

[17] Dooley, D., McGinley, B., Hughes, C., Kilmartin, L., Jones, E., & Glavin, M. A blind-zone detection method using a rear-mounted fisheye camera with combination of vehicle detection methods. IEEE Transactions on Intelligent Transportation Systems, 17(1), pp. 264-278, 2016.

[18] Magid, E., Tsubouchi, T., Koyanagi, E., & Yoshida, T. Static balance for rescue robot navigation: Losing balance on purpose within random step environment. In Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on, pp. 349-356, 2010.