External RGB-D Camera Based Mobile Robot Localization in Gazebo Environment with Real-Time Filtering and Smoothing Techniques

Kirill Kononov, Roman Lavrenov, Lilia Gavrilova, and Tatyana Tsoy

Abstract In previous work, we successfully studied the possibility of a mobile robot localization using an external RGB-D camera. We conducted virtual experiments in a Gazebo simulator using ROS with a Turtlebot3 Waffle Pi as a mobile robot. find_object_2d package was used to localize the Turtlebot3 and send its computed position to ROS. Thus, we extended our research and ratchet that result up by implementing the algorithm for filtering and smoothing the computed mobile robot position. Our task was to develop a thin library that is a wrapper over the find_object_2d package. The filtering algorithm and smoothing can compute the supposed robot position or predict it in cases when the robot disappears from the camera's field of view. We conducted virtual experiments over again and draw a comparison between previous results (without filtering and smoothing algorithm) and current results (using filtering and smoothing algorithm with slight improvements).

1 Introduction

Correct robot localization is important for any mobile robot tasks, including path planning, mapping [1], SLAM, performing critical tasks, operating in special environments, etc. [2–4]. In most cases, an inaccurate localization becomes a serious problem that might cause a wrong robot behavior [5]. An incorrect mobile robot localization is an considerable problem that arises due to accumulation of odometry errors, harsh environment, sliding and slippage of wheels on an underlying support surface, noisy or unstable GPS signal, and other different problems [6]. This way, a robot transmits less and less relevant data about its location over time.

Integration of filtering and smoothing methods into localization algorithms could significantly improve a localization accuracy [7]. In [8] Kalman filter [9] was used

K. Kononov · R. Lavrenov (✉) · L. Gavrilova · T. Tsoy
Laboratory of Intelligent Robotics Systems (LIRS), Intelligent Robotics Department, Institute of Information Technology and Intelligent Systems, Kazan Federal University, Kazan, Russia
e-mail: lavrenov@it.kfu.ru
URL: https://www.kpfu.ru/eng/itis/research/laboratory-of-intelligent-robotic-systems.com

A. Ronzhin and V. Shishlakov (eds.), *Electromechanics and Robotics*, Smart Innovation, Systems and Technologies 232, https://doi.org/10.1007/978-981-16-2814-6_20

for filtering and smoothing of a mobile robot localization based on odometric and sonar sensors. In [10] a Particle filter [11] was employed for smoothing robot self-localization, which was based on a strength of a WLAN signal.

In our previous work, we had studied a possibility of tracking a mobile robot position within the Gazebo simulator [12, 13] using an external RGB-D camera and find_object_2d package [14]. find_object_2d package [14] is a simple application that allows to detect a particular object within an image from a pre-created dictionary of objects using different types of OpenCV [15] detectors and descriptors. We employed ORB (Oriented FAST [16] and rotated BRIEF [17]) algorithm [18] since it provides a good combination of the detector and the descriptor for a task, which requires to localize the mobile robot rapidly and accurately. The resulting trajectory of the robot computed positions was not smooth and formed a chaotic zigzag line, the robot frequently failed to calculate its position (due to find_object_2d calculations' failures in about 20% of cases) and an average localization inaccuracy was 0.14 m while traveling within a 6 × 6 m room. A new algorithm presented in this paper performs filtering and smoothing of a moving robot localization data, successfully solves the trajectory smoothness issue, failures of a position calculation, and improves the average localization inaccuracy in approximately 2.25 times.

2 Related Work

The usage of filtering and smoothing algorithms for localization were described in [8] and [10]. In [8] Kalman filter [9] was used for filtering and smoothing of the mobile robot localization based on odometric and sonar sensors. In [10] the realization of Particle filter [11] was used for smoothing robot localization which is based on the strength of WLAN signal. In our work, we introduce an algorithm-helper that assist the robot to localize itself indoor using an external RGB-D camera. Thus, the developed method of filtering and smoothing of the mobile robot localization will increase its accuracy and ensure a smooth trajectory of its movement.

3 Proposed Approach

In our research we used the Gazebo simulator [13] with ROS [19] to simulate mobile robot movements in an arbitrary empty room. Without loss of generality was selected a room of size 6 ×6 m and was connected a RGB-D camera to the ceilings at a geometrical center of the room. This allows the camera to capture an entire floor surface and localize the robot. To track the robot position, find_object_2d package provided its coordinates through tf [20] and publish them in ROS [19]. Inaccuracy of localization was calculated as a translation between robot base_link coordinates and the obtained from find_object_2d coordinates. One of the issues that arose during virtual experiments was a piece-wise trajectory of the computed robot coordinates,

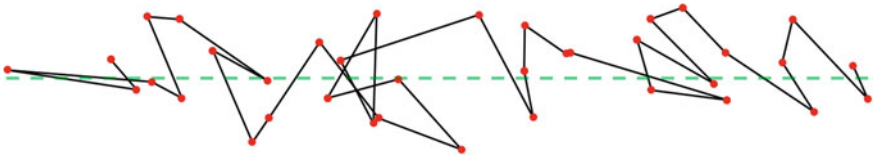

Fig. 1 An example of the robot coordinates trajectory computed with find_object_2d package [14]. Black dots are the coordinates, red line is the trajectory constructed from these points, and green dotted line is the real trajectory of the robot

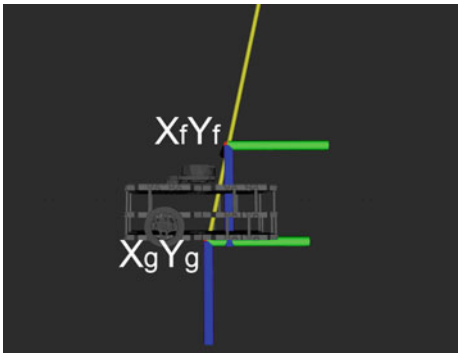

Fig. 2 XfYf frame origin corresponds to (x_f, y_f) point, $x_g y_g$ frame origin corresponds to (x_g, y_g) point

which was obviously incorrect (Fig. 1). A natural approach to this problem was to apply a filter that could smooth a trajectory and thus increase the accuracy of the localization.

Package find_object_2d computes robot position (x_f, y_f) as a closest to the camera point of the robot. In [21] obtained (x_f, y_f) was further projected orthogonally onto the XY plane of a support surface (floor) and next a distance from that projected point to base_link of the mobile robot was calculated. First improvement in the current approach is a straight line from the camera frame origin to (x_f, y_f). The line was further extended until it reached XY plane. The intersection point was labeled as (x_g, y_g) (Fig. 2).

Another improvement is switching the coordinate frame that we accepted as the correct robot position from base_link frame origin (in [21]) to base_scan frame origin. We use the base_scan as the closest coordinate frame to the robot center (Fig. 3). It was done due to the previously [21] calculated by us localization inaccuracy as the translation between computed robot position frame and base_link frame. The last one is placed between robot center and its front border. Thus, we use base_scan frame to calculate the inaccuracy of the localization, since it locates quite close to the robot center.

Before comparing the difference between localization accuracy before and after improvements, we want to explain the filtering algorithm we used in our previous work [21]. We introduced the α-value that means the distance between real robot position and robot position computed with find_object_2d [14]. If in a certain iteration

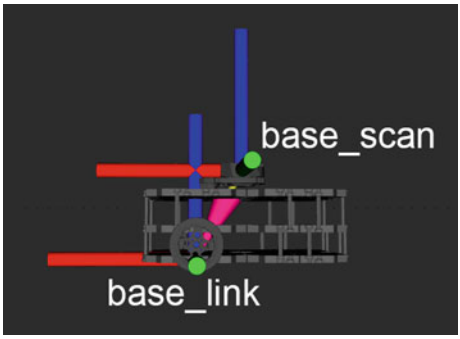

Fig. 3 Difference between positions of base_link and base_scan frames. base_scan frame is quite close to the robot center instead of base_link frame

α was more than 0.25 m, we would not accept this computed robot position. We undertook a small research and found out that such value of α is the most optimal and effective since it provides about 81–82% of acceptable robot position computations and the average value of inaccuracy was 0.13 m. Increasing the α value would also increase the percentage of acceptable robot position computations and as well it would increase the average inaccuracy of robot localization. Since we are interested in increasing the percentage of acceptable computed positions and decreasing the average computation inaccuracy, in our previous work we chose the α value equal to 0.25 m as the most effective [21]. However, those two new improvements make it possible to change the α value to get better results that are presented in Table 1.

These two improvements had a positive effect on the overall localization accuracy. Finally, our main improvement was the development of the algorithm which filters and smooths post-data from the find_object_2d package [14] and sends the processed predicted position of the mobile robot back to ROS [19]. We used tf package [20] to compare a predicted position with a real position, analyze received results, and to compare them to the previous research results.

Table 1 Comparison of the different configuration of the α value

Configuration	Acceptable computations (%)	AVG inaccuracy (m)
$\alpha = 0.25$ m (Before improvements)	82	0.14
$\alpha = 0.25$ m (After improvements)	97	0.09
$\alpha = 0.14$ m (After improvements)	82	0.075

The implemented improvements make it possible to obtain better results with the same value of α: acceptable computations increased from 82 to 97%, and the average inaccuracy decreased from 0.14 to 0.09 m. Reducing α to 0.14 m gives an average inaccuracy of 0.075 m, while saving percentage of acceptable computations at 82% as it was before the improvements

4 Algorithm Explanation

Developed filtering and smoothing algorithm uses data from find_object_2d package [14] which sends its data about a detected robot to ROS [19] through tf [20]. Then, our algorithm processes that data to compute the supposed position of the detected robot (Fig. 4).

Parameters are used in our algorithm:

- **window_size:** size of the array which contains the history of previous coordinates. The larger this value, the smoother the trajectory becomes, if so the algorithm reacts more slowly to sudden changes in speed and direction of the robot movement. Further denotes as "Window size".
- **alpha:** value for filtering input coordinates. If the distance between current and previous coordinates is more than this value, the current coordinate is not accepted and replaced by the predicted position which is calculated based on calibration. Further denotes as "α".

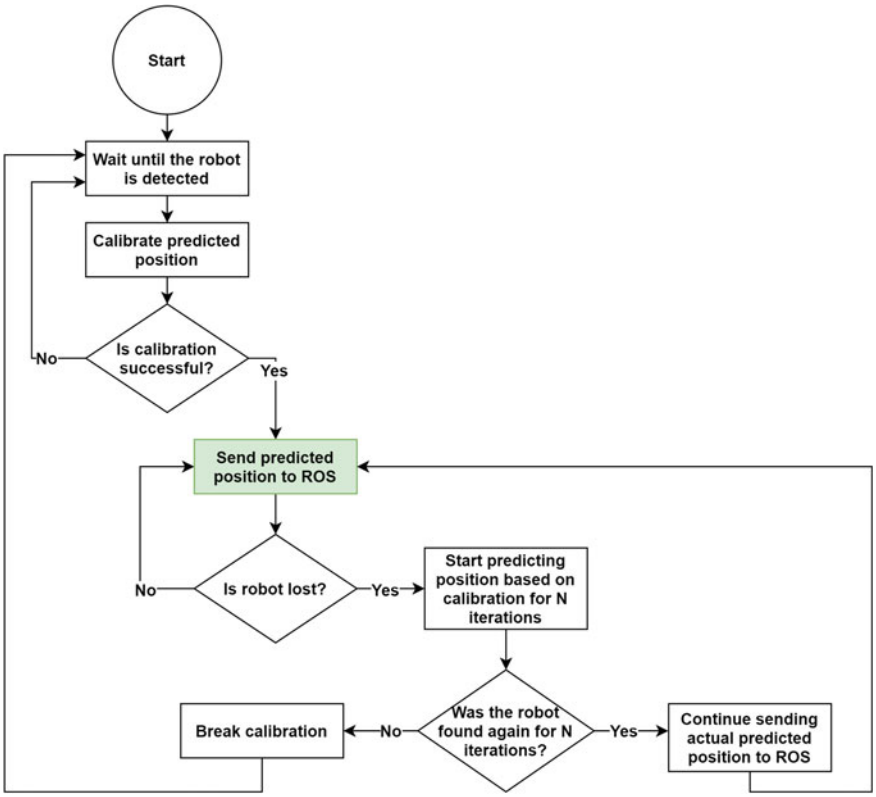

Fig. 4 Flow diagram which explains our filter and smoothing algorithm

- **calibration_threshold:** delay before the start of the calibration. This delay is necessary because the find_object_2d package [14] is very unstable to localize a robot that enters the camera's field of view when its body is not fully visible. Further denotes as "Calibration threshold".
- **frame_loss_threshold:** delay before resetting of the calibration. Used while the algorithm is predicting the movement of a robot that cannot be localized by the find_object_2d [14] package after a successful calibration decrements at each iteration if the robot was not detected. When this parameter reaches 0, the calibration is reset and the robot position prediction stops. Further denotes as "Frame loss threshold".

The developed algorithm contains the following parts: (a) calibration of predicted position, (b) filtering and smoothing the trajectory of robot movement based on calibration, (c) predicting the position of the robot that has disappeared from the camera's field of view.

Calibration of the predicted position is the process which tries to capture robot position for a certain count of iterations and then analyze received localization data. Analyzing of that data means the calculation of the average offset between computed coordinates (See Eq. 1):

$$\text{Average Offset} = \left(\sum_{i=2}^{\text{WindowSize}} (\text{Buffer}_i - \text{Buffer}_{i-1}) \right) / \text{WindowSize}, \tag{1}$$

where WindowSize specifies the size of the buffer. Buffer is the coordinate container of previously computed coordinates, i is the index of the coordinate in the buffer.

After the successful calibration, we receive average offset between successive coordinates and initialize a new instance of the "Predicted position" which contains $(X; Y)$ coordinate of the last robot position received from find_object_2d package [14]. At this moment, algorithm has an array-buffer of last X (X is equal to the Window size of the Buffer) computed coordinates of the robot. During the next iteration, we receive new computed robot coordinates from find_object_2d package [14] and delete the oldest element from the buffer. Further, we calculate average offset between all remaining elements in the buffer including the last computed one, add this offset to the predicted position coordinate and push this coordinate to the buffer (See equation to compute predicted position):

$$\begin{aligned} \text{Average Offset} = \Bigg(& \sum_{i=2}^{\text{WindowSize}-1} (\text{Buffer}_i - \text{Buffer}_{i-1}) \\ & + (\text{ComputedPos} - \text{Buffer}_{\text{WindowSize}}) \Bigg) / \text{WindowSize}, \end{aligned} \tag{2}$$

$$\text{PredictedPos} = \text{PredictedPos} + \text{AverageOffset}, \tag{3}$$

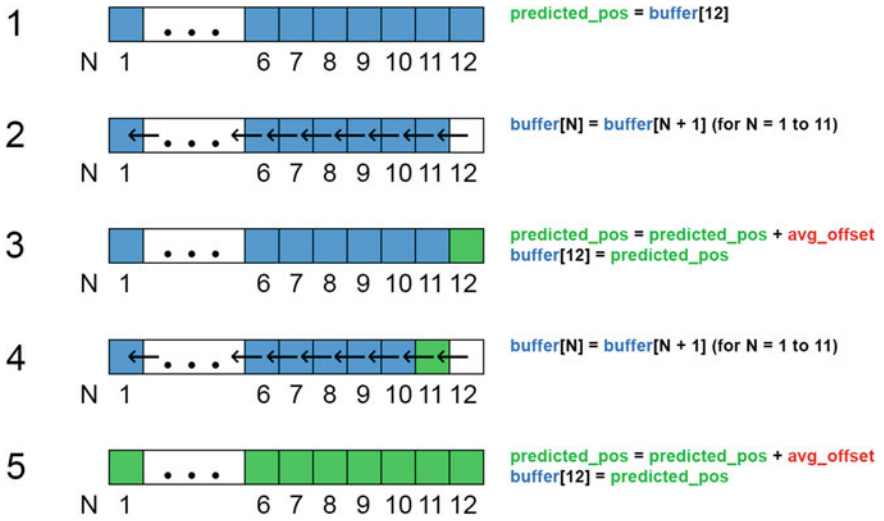

Fig. 5 Part of algorithm's work after successful calibration (Window size = 12). Blue cells are coordinates that are computed with find_object_2d package [14] and green cells are predicted (smoothed) coordinates. Step 1: set predicted position (predicted_pos) as the last coordinate from the buffer. Step 2: shift coordinates left by 1 (at this moment algorithm received a new computed coordinate). Step 3: calculate average offset (avg_offset) between coordinates in buffer (11 elements) including last computed coordinate (1 element). Add calculated avg_offset to predicted_pos and push updated predicted_pos in the end of the buffer. Step 4: repeat actions from Step 2. Step 5: after repeating Steps 3 and 4 for X iterations (X is equal to the Window size parameter) the buffer contains only smoothed coordinates

where WindowSize specifies the size of the buffer, Buffer is the coordinate container of previously computed coordinates, i is the index of the coordinate in the buffer, ComputedPos is the last computed position of the robot that is received from find_object_2d package [14].

These steps are repeated at every iteration and the algorithm produces a smooth trajectory of robot movement (Fig. 5).

Calibration could be failed in certain cases and our algorithm tries to predict the possible position of the robot:

- **Accumulated computation errors If find_object_2d** [14] computes the wrong robot position for a certain number of iterations in a row, the algorithm breaks, and the predicted position stops at a certain point. The distance between computed robot position and predicted position is greater than the α value, so our algorithm can't accept new computed coordinates due to this fact, and after a certain number of iterations, which is set by the value of the frame loss threshold parameter, calibration is canceled. Further algorithm waits for detection of the robot and begins a new calibration.
- **Robot loss from the camera's field of view** If the robot drives into the places where it cannot be detected by a camera, the algorithm tries to predict its possible

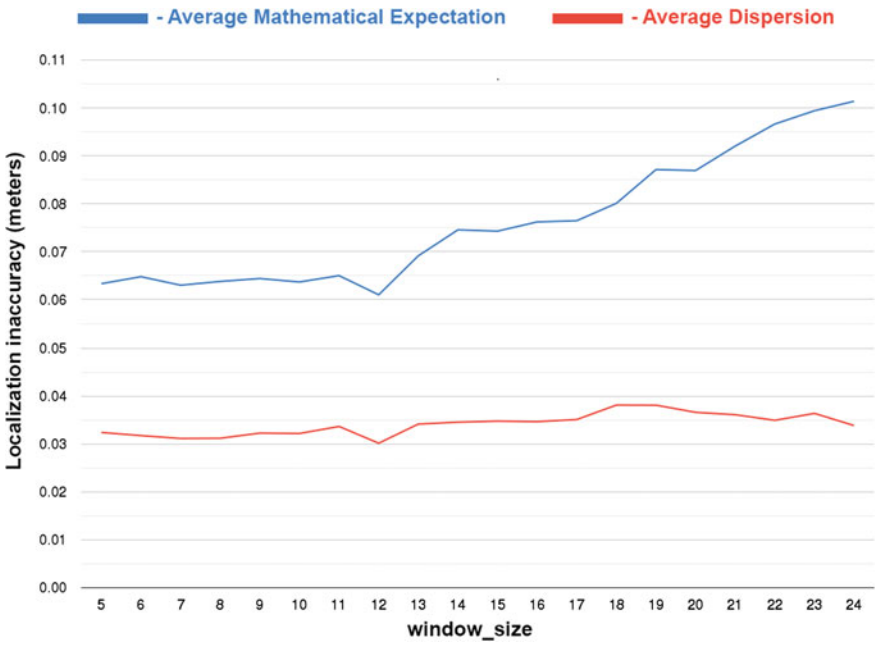

Fig. 6 Results of finding the most optimal Window size parameter for our algorithm. *X*-axis means the value of this parameter, *Y*-axis means inaccuracy of the robot localization in meters. Window size = 12 provides the most accurate filtering and smoothing results

movement trajectory for a certain number of iterations, which is set by the value of the frame loss threshold parameter. If the robot is not detected again, the calibration is canceled too and the algorithm waits for detection of the robot and begins a new calibration.

We conducted the experiments to define the most optimal and effective Window size and α parameters. Started with the first one (α was set at 0.25 m as the most optimal value in our previous work) we conducted 10 experiments for each value from 5 to 24, and found that 12 is the most efficient value for this parameter (Fig. 6), since it provides minimal average mathematical expectation and minimal average dispersion. So, we have found the most optimal value for the Window size parameter and further proceeded to determine the most optimal value for the α parameter. We repeated our experiments with Window size = 12, but changing α parameter from 0.05 to 0.5 with a step of 0.025. The experiments result show that all values that are more than 0.2 m give almost the same results (Fig. 7), but we decided to use α equal to 0.3 m.

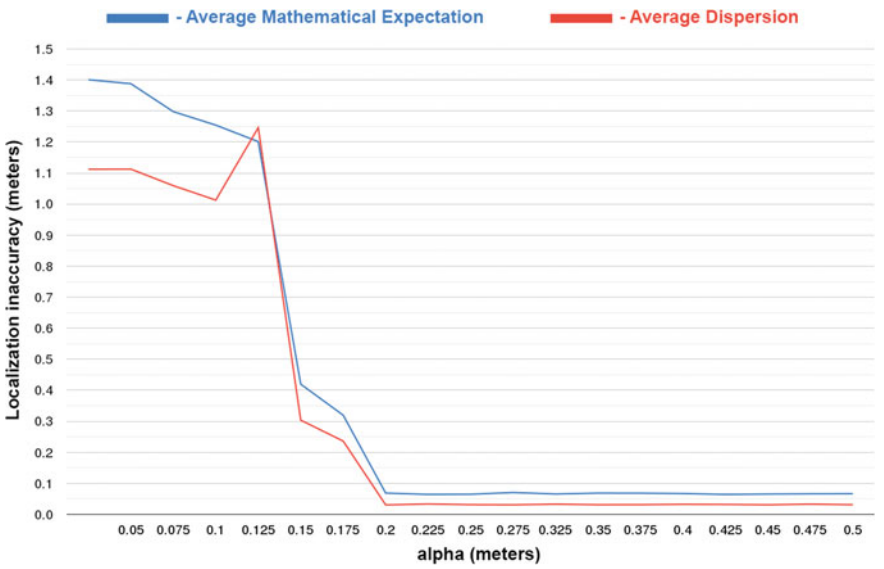

Fig. 7 Results of finding the most optimal α parameter for Window size parameter set at 12. X-axis means the value of the α parameter, Y-axis means inaccuracy of the robot localization in meters. With α greater than 0.2 m we get similar results

5 Experimental Results

We conducted the identical experiments as it was in our previous research [21] and compared old and new results for the same parameters. Our experiments contained linear and curvilinear routes:

- **Linear movement** 8 direct routes covering all areas of the room. Each route was tested 10 times to obtain average results.
- **Curvilinear movement** large circle (radius = 2 m) movement, small circle (radius = 1.25 m) movement, and 3 different chaotic routes. Each route was tested 10 times to obtain average results.

The analysis of the results (presented in Table 2) showed that using our algorithm for filtering and smoothing of the mobile robot localization data gives more accurate and stable average values.

The first method ("previously" in Table 1) supposes localizing of the mobile robot without using the developed filtering and smoothing algorithm. This method is based on a simple filter implemented on accepting computed coordinates only with an α less than 0.25 m (more details are available in our previous work [21]).

The second method ("currently" in Table 1) supposes the use of the new filtering and smoothing algorithm presented in this paper. The algorithm developed by us is based on the calibration of the predicted robot position. During the calibration, the algorithm calculates the average coordinate offset for certain iterations and then

Table 2 Comparison of the previous and current results obtained by experiments

	Mathematical expectation (m)	Dispersion (m)	Minimum (m)
Linear movement (previously)	0.136	0.045	0.028
Linear movement (currently)	0.044	0.015	0.010
Curvilinear movement (previously)	0.125	0.055	0.006
Curvilinear movement (currently)	0.072	0.029	0.030
Average value (previously)	0.130	0.049	0.017
Average value (currently)	0.058	0.022	0.020

All values calculated as the average of a specific type of experiment

uses this data to compute the smoothed predicted position of the robot. The average coordinate offset is updated at each new iteration, so the algorithm can react to sudden changes in the robot movements. Furthermore, the new algorithm can predict the approximate robot position in cases where the find_object_2d package [14] localizes the robot with significant errors or in other cases where the robot disappears from the camera's field of view (for example, there is some obstacle between the robot and the camera).

According to expectation, the linear movement gives the most accurate localization result, while curvilinear movement localization also becomes more accurate, but less accurate than linear movement localization. We can conclude that our algorithm reduces the localization inaccuracy by more than a half. However, the average minimum is still not close to zero, so we can confirm that our algorithm has a guaranteed error which is about 0.02 m.

6 Conclusions

It follows that the conducted experiments were successful and developed algorithm for filtering and smoothing the computed robot localization works properly. This method makes it possible to localize a mobile robot using the find_object_2d package [14] more accurately, avoid critical computational errors, successfully smooth the trajectory of the mobile robot movement, and predict the possible robot position when it drives into a place where the camera cannot detect it. The developed algorithm improves the average localization inaccuracy in approximately 2.25 times. Our future task is to develop a new method of mobile robot localization using an external RGB or RGB-D camera which will allow to detect and track several robots properly

in difficult tasks, for example during the localization of the identical robots or/and different robots using only one camera [22].

Acknowledgements This work was supported by the Russian Foundation for Basic Research (RFBR), project ID 19-58-70002.

References

1. Iakovlev, R., Saveliev, A.: Approach to implementation of local navigation of mobile robotic systems in agriculture with the aid of radio modules. Telfor J **12**(2), 92–97 (2020)
2. Giesbrecht, J.: Global path planning for unmanned ground vehicles. Technical Report Defence Research And Development, Suffield (Alberta) (2004)
3. Lavrenov, R., Magid, E.: Towards heterogeneous robot team path planning: acquisition of multiple routes with a modified spline-based algorithm. In: MATEC Web of Conferences. vol. 113, p. 02015. EDP Sciences (2017)
4. Magid, E., Tsubouchi, T.: Static balance for rescue robot navigation: discretizing rotational motion within random step environment. In: International Conference on Simulation, Modeling, and Programming for Autonomous Robots. pp. 423–435. Springer (2010)
5. Bai, Y., Wang, Y., Svinin, M., Magid, E., Sun, R.: Function approximation technique based immersion and invariance control for unknown nonlinear systems. IEEE Control Syst Lett **4**(4), 934–939 (2020)
6. Panov, A.I., Yakovlev, K.S., Suvorov, R.: Grid path planning with deep reinforcement learning: preliminary results. Procedia Comput Sci **123**, 347–353 (2018)
7. Ronzhin, A., Rigoll, G., Meshcheryakov, R.: Interactive Collaborative Robotics. Springer (2016)
8. Jetto, L., Longhi, S., Venturini, G.: Development and experimental validation of an adaptive extended Kalman filter for the localization of mobile robots. IEEE Trans Robotics and Automat **15**(2), 219–229 (1999)
9. Welch, G., Bishop, G., et al.: An introduction to the kalman filter (1995)
10. Nurminen, H., Ristimäki, A., Ali-Löytty, S., Piché, R.: Particle filter and smoother for indoor localization. In: International Conference on Indoor Positioning and Indoor Navigation. pp. 1–10. IEEE (2013)
11. Nummiaro, K., Koller-Meier, E., Van Gool, L.: An adaptive color-based particle filter. Image Vision Comput **21**(1), 99–110 (2003)
12. Abbyasov, B., Lavrenov, R., Zakiev, A., Yakovlev, K., Svinin, M., Magid, E.: Automatic tool for gazebo world construction: from a grayscale image to a 3D solid model. In: 2020 IEEE International Conference on Robotics and Automation (ICRA). pp. 7226–7232. IEEE (2020)
13. Koenig, N., Howard, A.: Design and use paradigms for gazebo, an open-source multi-robot simulator. In: 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566). vol. 3, pp. 2149–2154. IEEE (2004)
14. Labbé, M.: Find-Object. http://introlab.github.io/find-object (2011), Accessed 14 May 2020
15. Noble, F.K.: Comparison of opencv's feature detectors and feature matchers. In: 2016 23rd International Conference on Mechatronics and Machine Vision in Practice (M2VIP). pp. 1–6. IEEE (2016)
16. Viswanathan, D.G.: Features from accelerated segment test (fast). In: Proceedings of the 10th workshop on Image Analysis for Multimedia Interactive Services, London, UK. pp. 6–8 (2009)
17. Calonder, M., Lepetit, V., Strecha, C., Fua, P.: Brief: Binary robust independent elementary features. In: European Conference on Computer Vision, pp. 778–792. Springer (2010)
18. Rublee, E., Rabaud, V., Konolige, K., Bradski, G.: Orb: An efficient alternative to sift or surf. In: 2011 International Conference on Computer Vision, pp. 2564–2571. IEEE (2011)

19. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y., et al.: Ros: an open-source robot operating system. In: ICRA Workshop on Open Source Software. vol. 3.2, p. 5. Kobe, Japan (2009)
20. Foote, T.: tf: the transform library. In: 2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA), pp. 1–6. IEEE (2013)
21. Kononov, K., Larvenov, R., Tsoy, T., Martínez-García, E.A., Magid, E.: Virtual experiments on mobile robot localization with external smart RGB-D camera using ROS. In: IEEE Eurasia Conference on IOT, Communication and Engineering (2021)
22. Safin, R., Garipova, E., Lavrenov, R., Li, H., Svinin, M., Magid, E.: Hardware and software video encoding comparison. In: 2020 59th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE). pp. 924–929. IEEE (2020)