

# We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

6,100

Open access books available

149,000

International authors and editors

185M

Downloads

Our authors are among the

154

Countries delivered to

TOP 1%

most cited scientists

12.2%

Contributors from top 500 universities



WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index  
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?  
Contact [book.department@intechopen.com](mailto:book.department@intechopen.com)

Numbers displayed above are based on latest data collected.  
For more information visit [www.intechopen.com](http://www.intechopen.com)



Chapter

# Scalable Algorithms for Simultaneous Mapping and Localization of Mobile Robot Swarms

*Anton Filatov and Kirill Krinkin*

## Abstract

The chapter is devoted to the development of scalable algorithms for multi-agent solution of the SLAM problem. These algorithms are applicable to robots with limited computational resources, having limited computational power and memory, small spatial size, and power from a portable battery. To simplify the description, only robots equipped with LIDAR are considered. The main focus is as follows: a scalable multi-agent SLAM algorithm based on Dempster-Shafer theory; an algorithm for filtering two-dimensional laser scans to free up computational resources; evaluation of the accuracy of the map and trajectory constructed by the multi-agent algorithm; performance evaluation on resource-limited computing devices.

**Keywords:** autonomous systems, mobile robots, artificial intelligence, localization, SLAM

## 1. Introduction

Service robots become more and more common every year. There are a lot of areas where they can be applied: medicine, i.e., COVID robots, delivery, rovers, etc. One of the tasks that a service robot faces is an orientation on the environment. The accuracy of the map in the onboard computer's memory, according to which the robot is moving, determines how precisely it will move. The determination of the robot's own position on this map is also significant. Usually it is necessary to determine its own position to an accuracy of centimeters, no satellite location system is capable of it.

To ensure accurate localization, algorithms that solve the problem of SLAM (Simultaneous Localization And Mapping) are used. If it is known that the environment may dynamically change, then the most reasonable approach is to build a map online instead of using a pre-constructed one. For example, a car driving on public roads must avoid potholes on the road or avoid temporarily blocked sections of road. Equipping an unmanned car with such information in advance is a challenge. So using a pre-constructed map is certainly helpful, but it is also necessary to update the map as you drive.

Using multiple agents working together and building a map of the environment together allows for both faster map building and more accurate localization for each robot individually. The acceleration is achieved due to the fact that the researched area is divided into sections, and each section is studied by only a subgroup of agents, and then all the studied sections of the map are combined. The increase in accuracy compared with the single-agent algorithm is achieved due to the fact that each agent not only complements but also verifies the map of other agents. This allows correcting possible errors that occur in calculating its own position and constructing the map on the fly.

The **goal** of this work is to develop scalable algorithms for multi-agent SLAM, applicable to robots with limited computational resources. To increase the accuracy of the map construction, it is proposed to use **the Dempster-Shafer theory** [1] instead of the classical Bayesian one. Experiments have shown that the use of this theory increases the accuracy of the SLAM algorithm. In addition, the developed algorithms can be applied to robots with limited computational resources. In the context of this paper, a robot is a mobile platform equipped with a computing device and a LIDAR—a sensor that can measure the distance to obstacles surrounding the robot. Resource constraints are applied to the computing device, the applicability of different LIDARs is not analyzed in this paper. Thus, a robot with limited computational resources is a robot with limited computational power and RAM, small size, and power from a portable battery. At the moment, typical representatives of such devices are Raspberry Pi or Jetson Nano products.

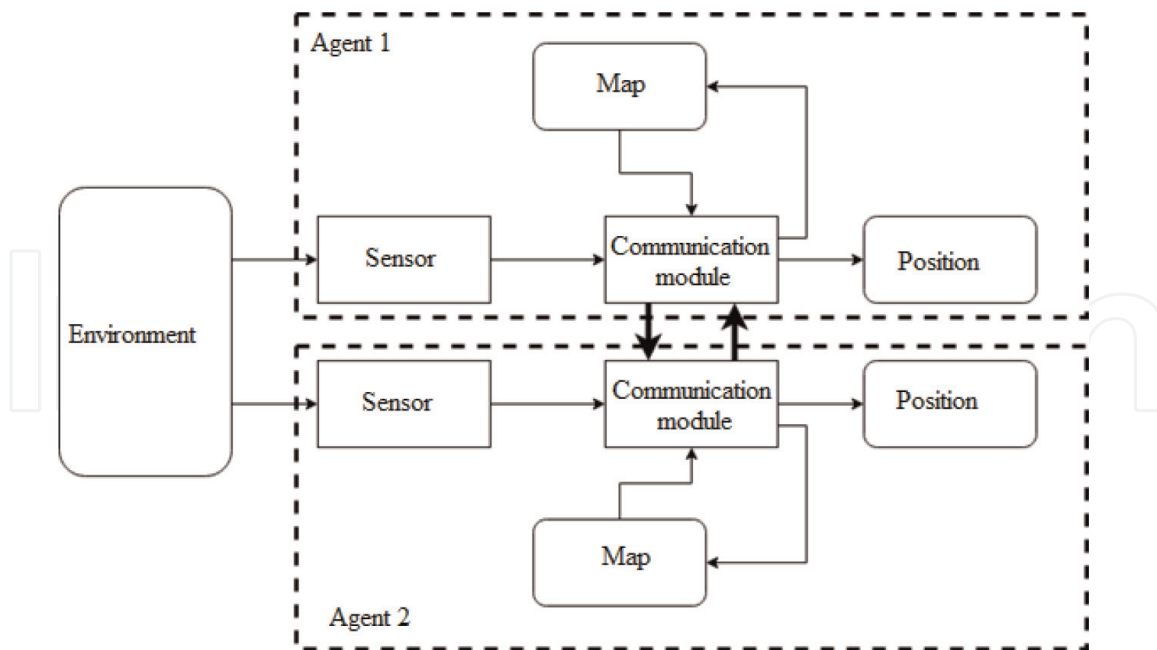
## 2. Swarm

First of all, it is necessary to introduce the notion of “swarm,” which is the executor of the multi-agent SLAM algorithm. A swarm of robots is a set of autonomous robots with comparable technical equipment for observing the environment in a limited area, they have no hierarchy and exchange information with each other to mark the environment together. Therefore, each robot in the swarm performs the same functions, there is no vertical hierarchy in the swarm. There is no central node—the server, whose absence could cause the entire system to stop working.

There are various papers describing the solution to the SLAM problem, both for a single robot and for a swarm. For example, in the works [2–4], the single-agent algorithms that are popular at the time of these works are listed and compared. There are algorithms that were directly developed as multi-agent algorithms. Usually their peculiarity is that there is some hierarchy and distribution of roles in the swarm. However, there are approaches where robots are independent agents and exchange data with each other asynchronously. In addition, there are a number of studies where single-agent pipelines were augmented with communication modules, and thus, the single-agent algorithm was “extended” to the multi-agent case. More details about the overview of such approaches are written in the papers [5–7]. In the current paper, it is considered the latter approach of those described above.

Each agent in the swarm independently determines the moment when it is necessary to start exchanging information with other agents. Naturally, the bases for such a decision are the same for all members in the pack. The model of this behavior for two robots in the swarm is presented in **Figure 1**.

The central node here is the communication module, which can perform a combination of its own data with another agent data. If communication with other robots in the swarm is not required, the communication module processes only its own data.



**Figure 1.**  
The model that describes the considered multi-agent SLAM algorithm.

### 3. Algorithm description

The developed algorithm will be used by the agents in an indoor environment. This means that the size of the studied space does not exceed several hundred square meters. In this case, it is impossible to use GPS, because the error of positioning using GPS is several meters, which is unacceptable. One can use a LIDAR as a sensor, the effective radius of which is measured in tens of meters. Choosing a LIDAR as a sensor allows you to build a map in the form of a grid of occupancy. Thus, the map constructed as a result of the algorithm will be a plan of the building on which the agents move.

Important fact is that the problem of controlling the mobile platform to explore the environment is not considered. The task of the developed algorithm includes only the processing of measurements obtained from the LIDAR and the odometry sensor and the construction of the map.

The indoor environment in which agents move should not contain dynamically changing areas. The appearance of rare and small noises is possible (e.g., the movement of people or other mobile platforms). However, moving furniture or columns in the room, especially at times when the agent is not observing the relevant part of the environment, is not allowed.

It is mandatory that all agents executing the SLAM algorithm must be equally equipped: not only have the same computing capabilities, but also the same mobile platform and the same LIDARs. This requirement is due to the fact that each agent executes the same algorithm, so the output data structure of the agents must be the same. If, for example, the computing power of one of the agents is greater than that of the other, the quality of the map generated by the agent with less power will be worse due to the lack of time for data processing. In this case, the map obtained as a result of the merging will not only contain areas with different degrees of clarity, but if some areas intersect, the less accurate map can introduce a significant error in the result.

### 3.1 Description of the features of the multi-agent algorithm core

In order for the algorithm to work robustly, it is necessary to get laser scan data and odometry data as inputs. It has already been said that odometry is not a mandatory parameter; however, it serves to increase accuracy.

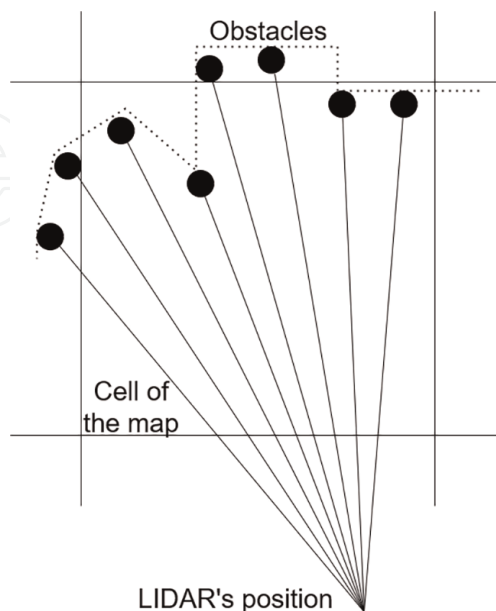
Odometry (as a prior estimation of position), together with the new laser scan and the currently constructed map, is passed to the input of the scan matcher [8].

The task of this module is to calculate the difference between a prior position estimation and the true position. The true position is the one from which a given laser scan can be captured. A laser scan may contradict the map due to errors of the scan matcher in previous steps or due to an incomplete map; the goal is to compute the most likely scan position.

To calculate the most likely position, one can use a Bayesian approach to probability calculation. The probability of a position is then calculated as the average sum of the probabilities of all scan points overlaid on the map from a given position. Each point of the scan denotes an obstacle, and under ideal conditions each point should be located in an occupied cell of the map. However, it is not enough to introduce a binary division into occupied and unoccupied cells. In reality, a map cell may be too large, and the scan points may be placed in it, as shown in **Figure 2**.

It is logical to assume that the cell has some probability of being occupied. This probability is based not only on the fact that at least one point of the laser scanner hits the cell, but also on how many beams from the LIDAR can pass through the cell before hitting an obstacle. This cell structure opens up the possibility of calculating the probability of position. The position probability can be calculated as the average sum of the probabilities of the cells in which the laser scanning points hit:

$$p_{pose}(scan) = \sum_{p \in points} o_p \cdot \omega_p \quad (1)$$



**Figure 2.**

*An example of a part of a laser scan. A lot of scanning points fall on a cell, but such a cell cannot be considered occupied.*

where  $p$  is a scan point,  $o_p$  is an occupancy of the point  $p$ ,  $w_p$  is a weight of the scan point.

The weight of the scan point is introduced to make the algorithm more stable in corridor conditions. In this case, different robot positions along the corridor direction have very close probability values. Differences in the values of probabilities are introduced by scan points that are different from the corridor walls. Usually such points are located in the direction of movement of the mobile platform. Consequently, laser scanning points located directly opposite the laser rangefinder have a greater weight.

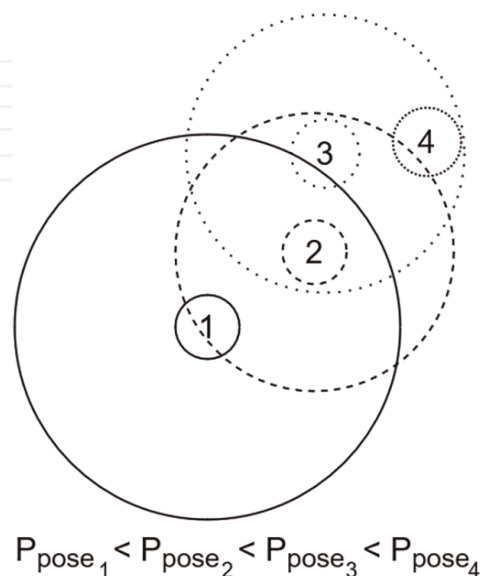
The scan matcher's goal is to calculate the position with the highest probability (when the laser scan points released from that position hit the squares on the map that have the highest probability of being occupied).

The approach to calculating such a position can be arbitrary. In this paper, the Monte Carlo scan matcher [9] is taken as the basis, the idea of which is a stochastic search of positions. Once a position with higher probability than all previous positions is found, the search begins in a lower radius around that position. This process is shown in **Figure 3**.

**Figure 3** shows the sequence of steps. Position 1 is a prior estimate of the position. With some radius around it, the positions are being chosen stochastically and their probabilities are calculated. The enumeration is carried out until position 2 with a higher probability than the probability of position 1 is found. The radius around position 2 is smaller than the radius around position 1. Then the operation is repeated several times. Practical application of this algorithm shows that it is sufficient to set a threshold for the number of calculations of position probabilities in general, instead of for the number of such positions. A guarantee of finding a solution by such an algorithm exists only if there are no distinct local maxima of position probabilities in the stochastic search area. Otherwise, it is necessary to increase the search radius, as well as the number of positions to search.

### 3.2 Applying Dempster-Shafer theory to increase the accuracy of the algorithm

According to the Bayesian approach, each cell in the map contains a number from 0 to 1 that determines the probability of a cell of being occupied. To calculate this



**Figure 3.**  
 The visualization of an iterative algorithm for stochastic search for a position with the highest probability.

probability, it is needed to estimate the position of specific scan points in the cell relative to the point from which the laser scan was observed. Each cell on the map has probability  $p$  of being occupied and probability  $q$  of being free. These probabilities are related by the equation  $p + q = 1$ . However, under real-world conditions, the “unknown” state should also be introduced. The cell may have such a state when it is in an unexplored region. Also, the cell may be in the “unknown” state when two consecutive scans contradict each other.

Thus, a cell is described by three probabilities: the probability of being occupied  $p$ , the probability of being free  $q$ , and the probability of being unknown  $u$ . These three probabilities are related by the identity  $p + q + u = 1$ . Unlike the previous approach, where one probability is expressed through another, this approach allows each probability to be expressed through a pair of others. The question arises how to combine cells following these rules.

To answer this question, the Dempster-Shafer theory is applied [1, 10]. To strictly follow this theory, it is also necessary to introduce a state of conflict in the cell of the map. From a natural point of view, conflict is the state of a cell not being in any of the possible states. But in reality, no cell in the map can be in a state of conflict, which means that this probability must be distributed among the other states. It should be noted: if a cell is in an unknown state, then it is simultaneously in free and occupied states.

In the context of the problem at hand, the Dempster-Shafer theory is applied during the scan matcher, when it is necessary to find out the probability of a map cell being occupied. Immediately after the scan matcher runs, the second step of applying this theory is to place the probabilities of being free and occupied into the empty map cells that have appeared in the LIDAR observation area. To do this, the rule of combining probabilities, described by the equation below, is applied.

$$m_{1,2} = \frac{1}{1 - K} \sum_{B \cap C = A \neq \emptyset} m_1(B) \cdot m_2(C) \quad (2)$$

where  $m$  is the probability of the state,  $A, B, C$  are the different states: whether the cell is free, occupied, or unknown (free or occupied),  $K$  is calculated by the formula

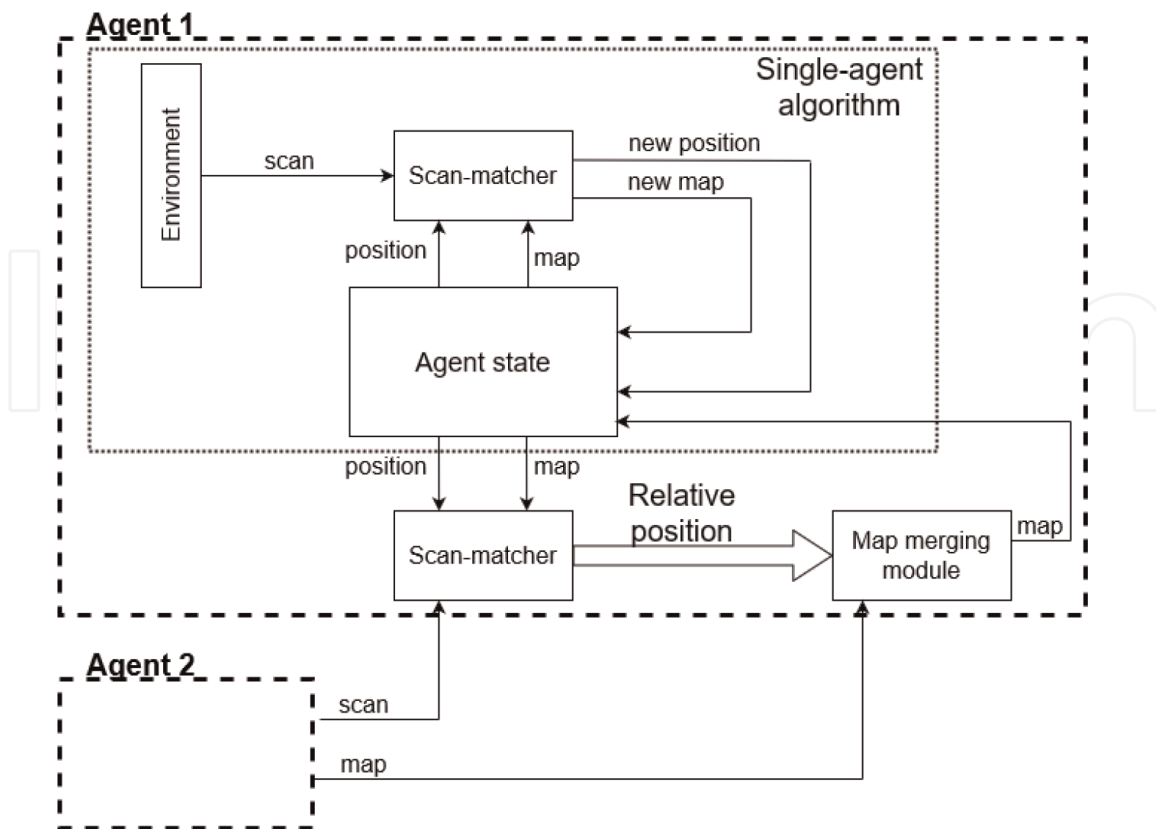
$$K = \sum_{B \cap C = \emptyset} m_1(B) \cdot m_2(C) \quad (3)$$

### 3.3 Description of the multi-agent SLAM algorithm

The distinctive feature of the proposed algorithm is that each agent has no knowledge of the other agents at any time. This allows the system to remain operational when one or more agents fail, or when only one agent remains in the system. Information is exchanged between agents only when two agents are in close proximity to each other.

Based on the described methods and hypotheses, one can construct an algorithm that solves the problem for a swarm of robots. The algorithm is an extension of the single-agent laser single-hypothesis algorithm, which uses an occupancy grid as a map. Each cell, in addition to the probability of being free, also contains a probability of being occupied and of being unknown. These probabilities follow Dempster-Shafer theory. The scheme of such a multi-agent algorithm is shown in **Figure 4**.

Consider the algorithm from the point of view of one particular agent, which solves the SLAM problem. The agent possesses only its own computational resources,



**Figure 4.**  
 The scheme of the multi-agent SLAM algorithm.

which are aimed at constructing a map and determining its own position on it. When another agent enters the field of view of the first agent, they begin the process of exchanging all the knowledge accumulated during the past time. It needs to be determined that the other agent has entered the field of view without using sensor data. Otherwise, it would be necessary to label the robot to distinguish it from its environment. And since the algorithm must operate in a situation with no prior knowledge of the environment, the approach that requires labeling the robot is not applicable.

Consequently, it is necessary to determine that one agent is in the direct line of sight of the other by means of some separate sensor. The choice of such a sensor is outside the scope of this work, but as an example, one can use the Bluetooth data transmission technology, the range of which is just a few meters. You can also use the wifi signal and based on the strength of the signal determine the distance between agents. It is possible to equip agents with radio sensors and calculate the distance between them using the radio signal.

When the agents are in close proximity, they can begin the process of exchanging the maps they have built. These maps definitely have a common part, since the data transfer takes place when the agents are almost in the same position and observe the same part of the environment. In addition to the maps, the agents pass each other the current laser scan, which allows applying a scan matcher algorithm that is similar to the one running in each agent's core in a SLAM algorithm.

Thus, an agent receives a laser scan from another agent. Using the scan matcher, it overlays this scan on its own map and determines how far apart the agents are in relation to each other. Consider in more detail the applicability of the scan matcher algorithm to determine such relative position. In general, the scan matcher takes as input the prior agent position, the current laser scan, and the constructed map. The



output parameters for calculating relative position are the vector by which a prior position estimate must be changed in order to calculate its posterior estimate. In other words, the scan matcher calculates the difference between the position obtained from the input and the position from which the resulting laser scan can actually be seen, given the resulting map.

This description of the scan matcher assumes that when it receives a laser scan from another agent as input, it will get the distance between the agents as output. The condition that the part of the environment captured on the scan is present in the map must be satisfied. The fulfillment of this condition is guaranteed. The scan matcher algorithm will be discussed in more detail in a further analysis.

After determining the mutual position of the agents, it is possible to start merging the maps. To do this, the second agent passes the first constructed map. Provided both maps obtained are error-free, map merging is a straightforward operation, where it is sufficient to go through each cell of both maps and choose either the largest, smallest, or average occupancy of the corresponding cells. The nature of the algorithm embedded in the core of each agent allows errors to occur during the map construction process that will never be corrected in the future. In particular, a single-agent SLAM algorithm may be not multi-hypothesis nor graph-based. Then the map constructed by one agent or the other may contain errors. Nor is it known whether the same error is contained in only one map or in both maps. To avoid this, the Dempster-Shafer theory is used.

The pseudocode of the multi-agent SLAM algorithm is presented below.

```

function MATCH(observation, pose, map)
  for random position  $\in$  neighborhood(pose) do
    pose_score  $\leftarrow$  try_insert(position + observation, map)
  end for
  return position[best_score], best_map
end function
while True do
  posei, mapi  $\leftarrow$  MATCH(observationi, posei-1, mapi-1)  $\blacktriangleright$  a single SLAM routine
  if another_robot_is_near() then
    foreign_mapi, foreign_observationi  $\leftarrow$  receive_foreign_state()
    foreign_pose  $\leftarrow$  MATCH(foreign_observationi, posei, mapi)
    relative_pose  $\leftarrow$  posei - foreign_pose
    mapi  $\leftarrow$  combine_with_thDS(mapi, foreign_mapi, relative_pose)
  end if
end while

```

#### 4. Laser scan filtering algorithm

LIDARs, as input data providers, have a common flaw: they collect too little and too much data at the same time. On the one hand, there is little data, because it is impossible to smooth or approximate it without significant loss of accuracy; on the other hand, there is a lot of data, because it takes a lot of memory to store and process data from modern laser scans, which are taken more than 30 times per second [11].

The need to develop a filter for laser scans arises. Normally one does not need to shoot laser scans as often, unless the scanner is mounted on a vehicle traveling at 60 km/h. In this case, the environment can change dramatically in 0.03 seconds. On the other hand, if the robot is moving indoors and has an average velocity of about 0.5–1 m/s [12], [13], such a

number of dense point clouds from the laser rangefinder is redundant. The idea of the filter is based on storing several consecutive scans in a sliding window and comparing each new scan with the scans from that window. If each new scan correlates strongly with each scan from the window—it should be discarded.

The basic idea of the developed algorithm is to compare the current laser scan with the previous one. If a new scan is similar to a previous one, it should not be processed; and in order to avoid noise in the observations, the current scan should be compared with several previous scans instead of just one. Thus, a sliding window of scans appears, which is capable of evaluating each new incoming scan.

Usually a laser scan consists of several thousand points. Calculating the correlation of scans by brute force method requires  $O(n^2)$  operations, which may exceed a million iterations. To reduce this amount, special points on the scan can be singled out, which would take a lot of time. Instead of using the raw laser scan data to calculate the correlation, it is proposed to build a histogram for each scan.

Consider the method of constructing a division histogram by distance ranges. For each scan, the maximum and minimum values of the distance to obstacles are known. Consequently, it is possible to divide this range spread into several interval, and then calculate the number of points corresponding to each interval. Two consecutive scans usually should not differ significantly, so the distance histograms should be close to each other. In practice, if the robot is not rotating, the difference between the two scans is insignificant, and the values of each column of the histogram change little. If the robot is rotating, the difference is more significant. However, you can update the approach: instead of calculating the number of points in each column, you can calculate the median value of distances for each interval. In this way, the two consecutive histograms become more diverse.

#### 4.1 Correlation criteria

The next step after creating the histograms of each scan is to calculate their correlation. Here one can use the methods of mathematical statistics and consider the histogram as a random variable with an unknown distribution. Since all the histograms from the window are generally similar to each other, it is assumed that the distribution is the same.

There are several well-known ways to calculate the correlation of random variables: Pearson correlation [14], Spearman correlation [15], and Kendall correlation [16]. The simplest is the Pearson correlation coefficient. It is calculated by the equation

$$P_{X,Y} = \frac{cov(X, Y)}{\sigma_X \sigma_Y} \quad (4)$$

where  $X, Y$  are some random variables,  $cov$  is covariance of random variables,  $\sigma$  is the variance of a random variable.

If a random variable consists of  $n$  observations, and  $x_i$  is the observed value of this variable at the  $i$ -th step, the Pearson coefficient can be calculated by the following equation:

$$P_{X,Y} = \frac{n(\sum_{i=1}^n x_i y_i) - (\sum_{i=1}^n x_i)(\sum_{i=1}^n y_i)}{\sqrt{[n\sum_{i=1}^n x_i^2 - (\sum_{i=1}^n x_i)^2][n\sum_{i=1}^n y_i^2 - (\sum_{i=1}^n y_i)^2]}} \quad (5)$$

The value of this coefficient ranges from  $-1$  to  $1$ . A value of  $1$  is a complete positive linear correlation,  $0$  is no linear correlation, and minus  $1$  is a complete negative linear correlation. This correlation is called linear because of the following geometric interpretation. Place the value of the first variable on the abscissa and the value of the second variable on the ordinate of the graph. If the points with the resulting coordinates belong to the same line with a positive derivative, then their correlation is  $1$ . If the derivative is negative, then the Pearson coefficient is minus  $1$ . If no line can be drawn, then the coefficient is  $0$ .

Kendall and Spearman coefficients are used to measure the ordered relationship between two measured variables [17]. It is a measure of rank correlation: the similarity of the rank of the data when ordered by each of the variables. The Spearman coefficient is defined as the Pearson coefficient between ranked variables.

The Kendall correlation coefficient is calculated as follows:

$$P_{X,Y} = \frac{\text{number of concordant pairs} - \text{number of discordant pairs}}{\binom{n}{2}} \quad (6)$$

To summarize, there are three well-known approaches to calculating correlation. The main drawback of the Kendall coefficient is algorithmic complexity. It requires calculating the rank of a random variable and then calculating the number of matched pairs. In the worst case, this may require  $N \log(N)$  operations. The Spearman coefficient is less complex, but it also requires the introduction of an order relation. Since correlation is computed for histograms of consecutive scans, correctly ranking the values in the histograms is a difficult task. For histograms that are similar in general, every small variation in values must be captured. Consequently, the order function must be sensitive to these fluctuations and at the same time show the true correlation. Therefore, the Pearson correlation coefficient is the most appropriate for the algorithm in question. Its complexity is  $O(n)$ , it does not require an order function, and it is sufficiently sensitive to fluctuations in the values in the histograms.

#### 4.2 Parameters and constants in the scan filtering algorithm

There are four parameters in the proposed algorithm, the fine-tuning of which must be paid attention to:

- The number of columns in the histogram and, therefore, the number of laser scan points in each column;
- the size of the sliding window;
- $P_{pair}$  intra-window correlation threshold (how much the new scan should correlate with each scan in the window);
- discard threshold – value of total correlation of scan with scans in the window  $P_{common}$ .

These four parameters affect the number and nature of the filtered scans. The first is the number of columns in the histogram. All histograms considered have a common

feature: the more columns contained in the histogram, the more details are processed for each scan. Consider the LIDAR used in the MIT dataset, which captures laser scans consisting of about 1000 points. Dividing these points into 50 columns means that each group of points contains an average of 20 points. Dividing into 10 columns results in groups of 100 points each.

Despite the intuitive notion that the higher the sensitivity, the higher the accuracy, in real data high sensitivity can be the other way around. For example, if a moving object appears in the field of view of a laser scan, it inevitably leads to a difference in two consecutive scans. Moreover, each sensor contributes noise to the observations, and sometimes (at short distances) this noise can be misinterpreted as a difference between scans. The results of the experiments presented in Section 4.5 show that for 1000 laser scanning points with an angle of view of should be grouped into 15 or 30 columns.

Another important constant is the size of the window in which the previous laser scans are located. The correlation estimate of the current scan is equal to the product of the correlation value of each scan from this window and the current scan. Pearson's correlation coefficient is taken as the correlation value. It is obvious that if the robot with the laser scanner moves fast, then a large number of different scans in the window decreases the final correlation estimate of the new scan. This means that the higher the velocity of the robot, the fewer scans should be stored in the window.

There is the experimentally obtained equation that relates the window size to the average velocity of the robot. The average velocity here is the average distance, in centimeters, that the robot travels between two laser scan images. This equation is a heuristic and allows us to relate the scanning capture property (speed) and the filter property (the amount of information that should be in the window).

$$Window\_size = \frac{27}{v^2} \quad (7)$$

where  $v$  is an average velocity in centimeters.

To determine the effect of the window size, it is necessary to consider two parameters closely related to each other and to the window size. The first parameter is the threshold for the Pearson correlation coefficient for each pair of scans. The second is the total correlation coefficient, which is equal to the product of the coefficients. Since the Pearson correlation coefficient is calculated for two consecutive scans obtained with a small time difference, it is obvious that they are highly correlated on average. Therefore, the threshold for a pair of scans should be at least 0.95, or better, 0.98. After calculating the correlation coefficient of a new scan with each scan in the window, all coefficients must be combined. A well-known way to do this is to multiply them. For example, the threshold for a window containing five scans and a pairwise correlation of 0.98 is  $0.98^5 = 0.904$ .

### 4.3 Evaluation of the quality and accuracy of the laser scan filter

For testing the scan filter was included in the operation of two SLAM algorithms: vinySLAM [18] and Google Cartographer [19]. The filter determines whether the scan should be processed or discarded before it is passed to the scan matcher of each of the listed algorithms. Consequently, if the scan must be processed, the time required for filtering is added to the total processing time of the scan. Therefore, it is necessary to estimate the algorithmic complexity of the filtering process and then present the

The sequence	vinySLAM	vinySLAM filter	Cartographer	Cartographer filter	% dropped
2011-01-20-07-18-45	$0.062 \pm 0.004$	$0.078 \pm 0.004$	$0.131 \pm 0.058$	$0.139 \pm 0.041$	59
2011-01-21-09-01-36	$0.080 \pm 0.018$	$0.089 \pm 0.013$	$0.153 \pm 0.072$	$0.163 \pm 0.094$	56
2011-01-24-06-18-27	$0.096 \pm 0.007$	$0.111 \pm 0.013$	$0.183 \pm 0.015$	$0.181 \pm 0.014$	59
2011-01-25-06-29-26	$0.094 \pm 0.006$	$0.100 \pm 0.002$	$0.176 \pm 0.010$	$0.179 \pm 0.012$	63
2011-01-27-07-49-54	$0.170 \pm 0.019$	$0.121 \pm 0.006$	$0.248 \pm 0.014$	$0.251 \pm 0.007$	52
2011-03-11-06-48-23	$0.534 \pm 0.085$	$0.543 \pm 0.034$	$0.586 \pm 0.174$	$0.642 \pm 0.191$	58
2011-03-18-06-22-35	$0.090 \pm 0.020$	$0.090 \pm 0.003$	$0.130 \pm 0.025$	$0.119 \pm 0.017$	52
2011-04-06-07-04-17	$0.183 \pm 0.014$	$0.213 \pm 0.027$	$0.188 \pm 0.011$	$0.185 \pm 0.011$	51
2011-01-19-07-49-38	$0.305 \pm 0.174$	$0.289 \pm 0.181$	$0.188 \pm 0.004$	$0.189 \pm 0.005$	50
2011-01-28-06-37-23	$0.361 \pm 0.175$	$0.348 \pm 0.152$	$0.378 \pm 0.025$	$0.399 \pm 0.030$	47

**Table 1.**

RMSE values and percentage of dropped scans for vinySLAM and cartographer on MIT dataset.

results of applying such a filter to real MIT dataset [20]. It is also required to estimate the proportion of scans that can be discarded without loss of overall accuracy (**Table 1**).

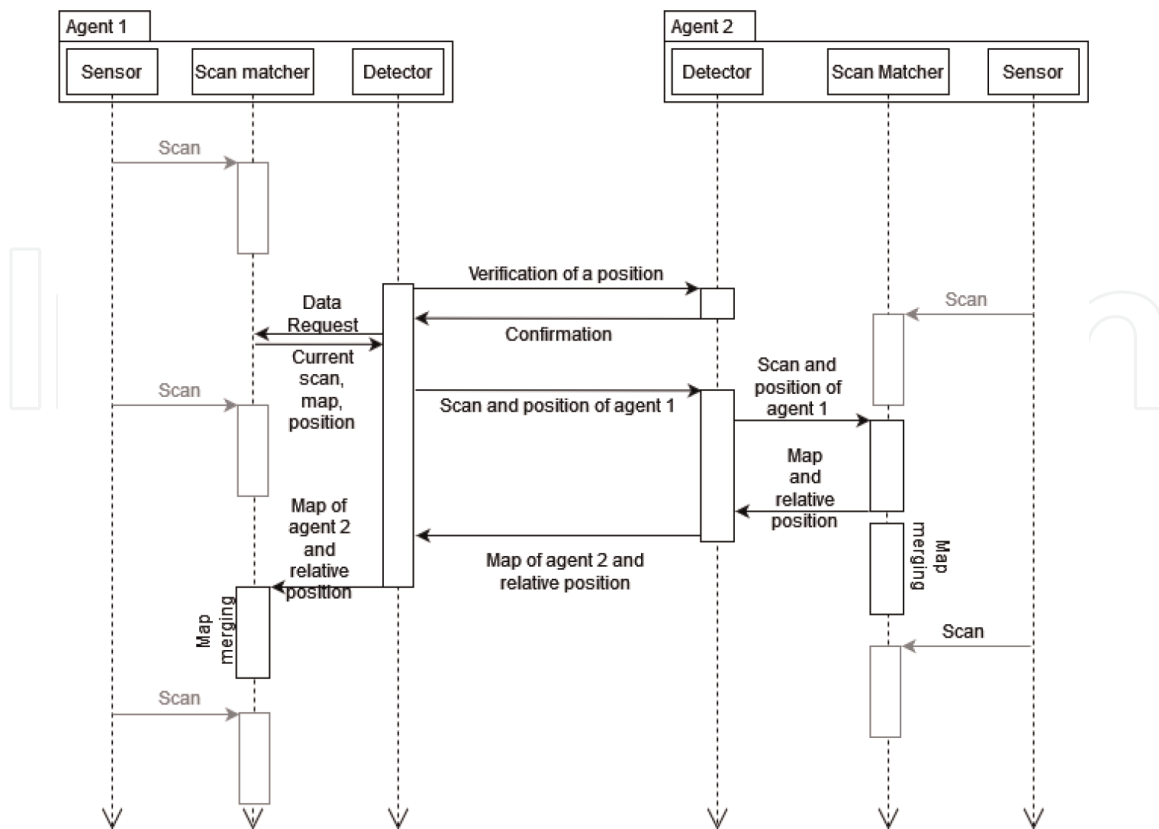
## 5. Accuracy measures of the multi-agent SLAM algorithm

The data flow exchanged between the main nodes in the developed software can be seen in the sequence diagram shown in **Figure 5**. For simplicity, this diagram shows only the two main nodes, the scan matcher and the detector. The other nodes are omitted, since interaction with them can be considered atomic operations that do not require additional actions. One can also assume that the scan matcher has all the information about the current map, the position of the robot, and the current observation.

In order to assess the accuracy of the SLAM algorithm, it is necessary to quantitatively compare the artifacts obtained during its operation: the map and trajectory of the robot with the true map and trajectory. Quantitative comparison of maps is a task of image analysis. The characteristics of such a comparison are quite comprehensive and require specific knowledge of the true map [21]: scale, permissible error in drawing the image, and others. It is also necessary to calculate what part of the true map the robot had time to observe.

Therefore, to evaluate the quality of the SLAM algorithm, a comparison of the trajectory is used as a sequence of positions obtained during the execution of the SLAM algorithm with the true trajectory. The datasets in question are not accompanied by a true trajectory. However, the MIT University dataset comes with a table that can be used as input to solve the localization problem (in other words, together with a utility that allows you to calculate the trajectory of the robot in the recorded dataset, based on the solution of the localization problem).

Since the true trajectory of the robot for each MIT dataset is known, namely the robot position and the timestamp corresponding to this position, it is possible to



**Figure 5.**  
 UML sequence diagram in developed software.

calculate the deviation at each time point. From such an array of deviations, the standard deviation is calculated as a characteristic of the quality of performance of the SLAM algorithm. The disadvantage of the proposed approach is the probability of error in calculating the true trajectory when solving the localization problem. However, it is the most reliable way to quantify the algorithm accuracy.

It is not enough just to calculate the standard deviation from the true trajectory, it is also necessary to determine how small it is. Therefore, a reliable algorithm is needed for solving the SLAM problem and comparing the standard deviation of different algorithms. As an estimating algorithm, gmapping [22] is often used or cartographer. In this paper, the comparison will be made with the cartographer algorithm, as well as with the vinySlam algorithm.

Thus, the first experiment to calculate the accuracy of the multi-agent algorithm consists of the following steps.

1. Run the algorithm on one of the data sequences from the MIT dataset emulating one robot from the swarm.
2. Run the algorithm on another data sequence from the same MIT dataset. You need to make sure beforehand that these sequences are written given the existence of a moment in time when the robots from both sequences are close to each other.
3. At the moment when the robots are close to each other, manually send a command to exchange the current observations and maps.

4. After exchanging data and updating maps, the agents continue executing the algorithm until the end of the recorded data sequence.
5. After completing the data set replay, each agent compares the trajectory it built with the trajectory built by the single-agent algorithm on the same data sequence.

The goal of experiment # 1 is to show that using another robot's map does not reduce accuracy compared with the single-agent algorithm and also allows the agent's map to be supplemented with areas it has not visited so far. This speeds up the work by matching the laser scan to an already constructed map, an algorithmically less costly operation than embedding the scan into an unknown map.

The second experiment consists of running the same sequence twice at the same time. One copy is played without change, and the second copy is played backward. This case guarantees that in the middle of the sequence there will be a point where both agents will be in at the same time. The sequence of steps in experiment 2 is as follows.

1. Run the algorithm on one of the data sequences from MIT.
2. Run the algorithm on the same sequence reproduced in the other direction.
3. When the robots are at the same point, send them a signal to exchange current observations and maps.
4. After exchanging data and updating maps, the agents continue executing the algorithm until the end of the recorded data sequence.
5. When the dataset is complete, each agent compares the trajectory it built with the trajectory built by the single-agent algorithm on the same data sequence.

The key feature of such an experiment is to demonstrate how using exactly the part of the map that the robot will move on in the future affects the accuracy of the results (**Tables 2 and 3**).

The results prove the accuracy of the proposed algorithm, since the RMS error is always less than 0.5 m and is almost always lower than that of the google cartographer graph algorithm. This result is mainly based on the accuracy of the single-agent

The sequence	Trajectory length, m	Multi-agent RMSE, m	Core RMSE, m	Cartographer RMSE, m
2011-01-20-07-18-45	76	0.045 ± 0.005	0.062 ± 0.004	0.131 ± 0.058
2011-01-21-09-01-36	87	0.080 ± 0.018	0.080 ± 0.018	0.153 ± 0.072
2011-01-24-06-18-27	87	0.096 ± 0.011	0.097 ± 0.007	0.183 ± 0.015
2011-01-25-06-29-26	109	0.094 ± 0.009	0.094 ± 0.006	0.176 ± 0.010
2011-01-28-06-37-23	145	0.395 ± 0.190	0.361 ± 0.175	0.201 ± 0.011
2011-01-27-07-49-54	94	0.167 ± 0.018	0.170 ± 0.019	0.248 ± 0.014

**Table 2.** RMSE values for experiment # 1 in comparison with RMSE of core SLAM algorithm and Google cartographer.

The sequence	Trajectory length, m	'Forward' RMSE, m	'Backward' RMSE, m
2011-01-20-07-18-45	38 + 24	0.044 ± 0.015	0.021 ± 0.005
2011-01-21-09-01-36	43 + 31	0.080 ± 0.011	0.068 ± 0.012
2011-01-24-06-18-27	43 + 41	0.106 ± 0.009	0.091 ± 0.003
2011-01-25-06-29-26	33 + 61	0.033 ± 0.013	0.097 ± 0.016
2011-01-28-06-37-23 part 1	41 + 39	0.184 ± 0.093	0.231 ± 0.040
2011-01-28-06-37-23 part 2	41 + 39	0.311 ± 0.187	0.272 ± 0.195
2011-01-27-07-49-54	31 + 62	0.142 ± 0.019	0.161 ± 0.022

**Table 3.**  
 RMSE values for experiment # 2.

version of the vinySlam algorithm. The largest error can arise if the result of calculating the mutual arrangement of the agents during map merging is a rotation error. Despite the fact that in this case the output map will be consistent, the output trajectory may differ markedly. It is fair to note that this problem occurs in any multi-agent algorithm; and even a graph algorithm can correct for it only under certain conditions, when the agents visit the most inconsistent part of the surrounding world several times.

In addition to estimating accuracy, it is necessary to determine the performance of the multi-agent algorithm on different hardware configurations. The goal of this experiment is to determine the applicability limits on low-performance computing devices. The performance measurements should be divided into two stages:

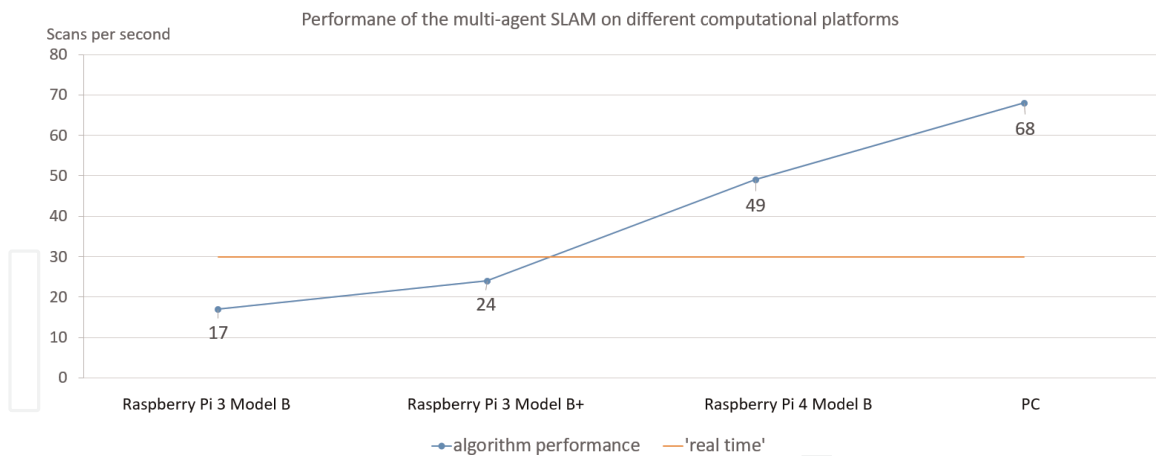
1. Determination of measurement processing rate during execution of the single-agent part of the algorithm
2. Determination of the rate of data exchange during the encounter, as well as the rate of mutual positioning and map updating

The performance measurements were made on the following configurations.

1. Raspberry Pi 3 Model B (Broadcom BCM2837 processor, 1GB LPDDR2 RAM, Ubuntu Xenial x64 operating system).
2. Raspberry Pi 3 Model B+ (Broadcom BCM2837B0 Quad Core 1.2GHz processor, 1GB LPDDR2 RAM, Ubuntu Xenial x64 operating system).
3. Raspberry Pi 4 Model B (Quad Core 1.5GHz Broadcom BCM2711 processor, LPDDR4 2GB RAM, Ubuntu Xenial x64 operating system).
4. Personal computer (Processor: Intel Core i7-860 4x2.8GHz, DDR3 8GB RAM, Ubuntu Xenial x64 operating system).

The evaluation was performed on Raspberry Pi computers [23], since they are inexpensive and very popular computing devices used in robotics. The experiment was also conducted on virtual machines with resources comparable to those of the Raspberry Pi. The results were too similar, so they are not presented in a separate





**Figure 6.**  
The amount of processed scans per second in different computational configurations.

graph. Therefore, it can be assumed that other boards with similar resources can be used instead of Raspberry products to obtain performance characteristics.

**Figure 6** shows the number of scans per second that can be processed on the listed configurations. As a conditional boundary is taken the input data processing frequency of 30 scans per second, which is comparable with the processing frequency of the human eye. Therefore, one can conventionally consider that if the algorithm is executed with a frequency of more than 30 scans per second, then it works in real-time mode.

## 6. Conclusion

According to the results of theoretical and experimental studies presented in this paper, the following conclusions can be made.

- Abandoning the graph structure along with the application of Dempster-Shafer theory allows us to achieve good performance of multi-agent SLAM algorithms running on low-powered robots. The rejection of role specialization in a swarm of agents provides good scalability and robustness.
- The application of a filtering algorithm based on the calculation of the correlation of the nearest two-dimensional laser scans allows increasing the frame processing speed up to 40%.
- The conducted experiments show the high accuracy of lightweight algorithms comparable with resource demanding algorithms, such as Google Cartographer. In particular, the root-mean-square error of the multi-agent algorithm was 9.4 cm at a distance of 100 meters covered by one agent, subject to a single synchronization with another agent. The error in determining the position of the robot is comparable to its dimensions.

The application of Dempster-Shafer theory can be promising for data filtering, leader election in graph-based algorithms, and for increasing the accuracy of combining maps with more than two agents.

## **Acknowledgements**

The authors would like to thank Saint Petersburg Electrotechnical University “LETI” and the Pavlov world-class research center in Personalized Medicine, High-Tech Medical Care, and Healthcare Technologies for provided support and materials for working on this paper.

## **Conflict of interest**

The authors declare no conflict of interest.


## **Author details**

Anton Filatov\* and Kirill Krinkin  
Saint Petersburg Electrotechnical University, Saint Petersburg, Russia

\*Address all correspondence to: [aifilatov@etu.ru](mailto:aifilatov@etu.ru)

## **IntechOpen**

---

© 2022 The Author(s). Licensee IntechOpen. This chapter is distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/3.0>), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited. 

## References

- [1] Dempster AP. The Dempster–Shafer calculus for statisticians. *International Journal of Approximate Reasoning*. 2008;**48**(2):365-377
- [2] Huletski A, Kartashov D, Krinkin K. Evaluation of the modern visual slam methods. In: 2015 Artificial Intelligence and Natural Language and Information Extraction, Social Media and Web Search FRUCT Conference (AINL-ISMW FRUCT). Helsinki, Finland. 2015. pp. 19-25
- [3] Krinkin K, Filatov A, Filatov A, Huletski A, Kartashov D. Evaluation of modern laser based indoor slam algorithms. In: 2018 22nd Conference of Open Innovations Association (FRUCT). Helsinki, Finland. 2018. pp. 101-106
- [4] Merzlyakov A, Macenski S. A comparison of modern general-purpose visual slam approaches. In: 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Prague, Czech Republic. 2021. pp. 9190-9197
- [5] Filatov A, Krinkin K. Multi-agent SLAM approaches for low-cost platforms. In: 2019 24th Conference of Open Innovations Association (FRUCT). Helsinki, Finland. 2019. pp. 89-95
- [6] Thrun S, Liu Y. Multi-robot SLAM with sparse extended information filters. In: *Robotics Research. The Eleventh International Symposium*. Siena, Italy: Springer; 2005. pp. 254-266
- [7] Kegeleirs M, Grisetti G, Birattari M. Swarm Slam: Challenges and perspectives. *Frontiers in Robotics and AI*. 2021;**8**:618268
- [8] Gutmann J-S, Schlegel C. Amos: Comparison of scan matching approaches for self-localization in indoor environments. In: *Proceedings of the First Euromicro Workshop on Advanced Mobile Robots (EUROBOT'96)*. Rome, Italy. 1996. pp. 61-67
- [9] Fox D, Burgard W, Dellaert F, Thrun S. Monte Carlo localization: Efficient position estimation for Mobile robots. *AAAI/IAAI*. 1999;**1999**(343-349):2-2
- [10] Yager RR. On the Dempster-Shafer framework and new combination rules. *Information Sciences*. 1987;**41**(2):93-137
- [11] Raj T, Hashim FH, Huddin AB, Ibrahim MF, Hussain A. A survey on LiDAR scanning mechanisms. *Electronics*. 2020;**9**(5):741
- [12] Blanc G, Mezouar Y, Martinet P. Indoor navigation of a wheeled mobile robot along visual routes. In: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. Barcelona, Spain. 2005. pp. 3354-3359
- [13] Paull L, Tani J, Ahn H, Alonso-Mora J, Carlone L, Cap M, et al. Duckietown: An open, inexpensive and flexible platform for autonomy education and research. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. Marina Bay Sands, Singapore. 2017. pp. 1497-1504
- [14] Benesty J, Chen J, Huang Y, Cohen I. Pearson correlation coefficient. In: *Noise Reduction in Speech Processing*. Berlin, Germany: Springer; 2009. pp. 1-4
- [15] Myers L, Sirois MJ. Spearman correlation coefficients, differences between. *Encyclopedia of Statistical Sciences*. 2004;**12**:1-10

[16] Abdi H. The Kendall rank correlation coefficient. *Encyclopedia of Measurement and Statistics*. 2007;2: 508-510

[17] Croux C, Dehon C. Influence functions of the spearman and Kendall correlation measures. *Statistical Methods & Applications*. 2010;19(4):497-515

[18] Huletski A, Kartashov D, Krinkin K. Vinyslam: An indoor slam method for low-cost platforms based on the transferable belief model. In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Vancouver, British Columbia, Canada. 2017. pp. 6770-6776

[19] Hess W, Kohler D, Rapp H, Andor D. Real-Time Loop Closure in 2d LIDAR SLAM. In: 2016 IEEE International Conference on Robotics and Automation (ICRA). Stockholm, Sweden. 2016. pp. 1271-1278

[20] Fallon M, Johannsson H, Kaess M, Leonard JJ. The Mit Stata Center dataset. *The International Journal of Robotics Research*. 2013;32(14):1695-1699

[21] Filatov A, Filatov A, Krinkin K, Chen B, Molodan D. 2d slam quality evaluation methods. In: 2017 21st Conference of Open Innovations Association (FRUCT). Helsinki, Finland. 2017. pp. 120-126

[22] Murphy K, Russell S. Rao-Blackwellised particle filtering for dynamic Bayesian networks. In: *Sequential Monte Carlo Methods in Practice*. Berlin, Germany: Springer; 2001. pp. 499-515

[23] Richardson M, Wallace S. *Getting Started with Raspberry PI*. Washington, D.C., USA: O'Reilly Media, Inc.; 2012