Evaluation of Modern Laser Based Indoor SLAM Algorithms

Conference Paper · May 2018

DOI: 10.23919/FRUCT.2018.8468263

5 authors, including:

Kirill Krinkin
Petersburg State Electrotechnical University
46 PUBLICATIONS **86** CITATIONS

Evaluation of Modern Laser Based Indoor SLAM Algorithms

Kirill Krinkin, Anton Filatov, Artyom Filatov
Saint-Petersburg Electrotechnical University "LETI"
St. Petersburg, Russia
kirill.krinkin@fruct.org, {ant.filatov, art32fil}@gmail.com

Artur Huletski, Dmitriy Kartashov
The Academic University
St. Petersburg, Russia
{hatless.fox, dmakart}@gmail.com

***Abstract*—One of the key issues that prevents creation of a truly autonomous mobile robot is the simultaneous localization and mapping (SLAM) problem. A solution is supposed to estimate a robot pose and to build a map of an unknown environment simultaneously. Despite existence of different algorithms that try to solve the problem, the universal one has not been proposed yet [1]. A laser rangefinder is a widespread sensor for mobile platforms and it was decided to evaluate actual 2D laser scan based SLAM algorithms on real world indoor environments. The following algorithms were considered: Google Cartographer [2], GMapping [3], tinySLAM [4]. According to their evaluation, Cartographer and GMapping are more accurate than tinySLAM and Cartographer is the most robust of the algorithms.**

I. Introduction

Simultaneous localization and mapping (SLAM) is the problem of building a map while at the same time localizing a robot within that map without prior knowledge of the map and the position. There are various SLAM approaches based on data from different types of sensors. A popular setup for mobile platforms provides:

- laser scan – a set of ranges from a platform to obstacles.
- odometry – information about platform's displacement;

The first type of input data represents a 2D top-down view of an environment that provides less information than a 3D view. So typically it requires less computations to be handled, which is more suitable for low-cost mobile platforms where computational resources are limited.

The SLAM problem is well defined but an algorithm that solves it for arbitrary environments has not been proposed yet [1]. Some implemented approaches are supposed to be used in an outdoor environment, some others – in indoor where straight lines predominate.

In spite of differences in 2D SLAM algorithms, they use common ideas and data structures. Information about an environment is stored as a certainty grid which is a two-dimensional array of identical units that at least keep their probability of being occupied. A platform position (robot pose) consists of coordinates and orientation. A combination of the robot pose and the corresponding map is a world state. The algorithms usually utilize a scan matcher component that corrects the robot pose estimated with a raw odometry by matching laser scan data with previous scans. A scan matcher can be implemented in multiple ways: bundle adjustment of scan points [5], a search of the best correlation between a entire scan and the map [6], etc.

Fig. 1 shows a high-level scheme of a typical laser scan based probabilistic SLAM method:

- $sensor_1$ and $sensor_2$ provide odometry data and laser scan respectively;
- *tracking* combines the odometry with the previous robot pose estimation;
- *fMap* converts a given scan to a robot pose (usually such conversion is done with a scan matcher taking into account already built map);
- *sensor fusion* merges two pose estimations;
- $\oplus$ adds a laser scan to the map.

The scheme specifies the algorithms based on a prediction-correction loop that consists of the following steps: a prediction of a robot pose (*tracking*), a correction with the data from exteroceptive sensors (*fMap* + *sensor fusion*) and a map update ($\oplus$).

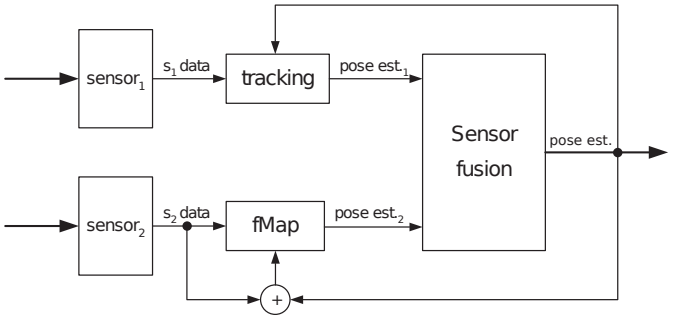


Fig. 1. The high-level scheme of a laser scan based SLAM

The following ideas could be found in popular 2D SLAM methods:

- single world state hypothesis tracking;
- multiple world state hypotheses tracking;
- a world representation with a graph.

The world state can be viewed as a random variable with an unknown probability density function (PDF) that is updated whenever new sensor data become available [7]. The plain single hypothesis tracking implies estimation and update of only the most probable world state. The tinySLAM [4], L-SLAM [8], FastSLAM [9] algorithms are examples of this

approach. The technique is not general since an environment may contain patterns which leads to a PDF with several maximums. If the map is stored as a grid a position of a scan already inserted into it cannot be changed afterwards.

The multiple hypotheses tracking keeps several states (based on grid maps), so there is an ability to maintain a set of probable maps until the PDF became unimodal. For example, GMapping [3] and DP-SLAM [10] implement this idea with a particle filter. It updates each world state (particle) independently by the prediction-correction loop described above. The aim of the particle filter is to approximate the PDF with a set of elements (world states) from its domain. The set has to be updated periodically with the resampling operation to reflect changes in the PDF. It populates a new set of particles in the neighborhood of the most probable states. However, resampling suffers from a particle depletion problem: there is a chance of the actual world state rejection. For example, to deal with this problem GMapping tries to make resampling rarely. Another strategy is to add new particles without considering existing ones.

In essence, the multiple hypothesis tracking keeps several rigid configurations of collected laser scans to provide diversity of maps. Another technique is to store a map as a graph that allows to modify scans' spatial constraints to achieve consistency [11].

The graph could be structured as follows: vertices contain the combination of a robot pose and an observation, edges are transformations between vertices. Such representation allows to correct a transformation (provided by a scan matcher) later during graph optimization. A map should be topologically correct in order to be accurately refined by this optimization. Topological correctness can be violated if a robot fails to detect itself in an already visited vertex (the loop-closure problem). The optimization is eventually performed when a new loop is closed in a graph. The aim of this process is to reduce and distribute the accumulated error among edges. The g2o library [12], for example, can be used to perform the optimization. The examples of methods that use graph map are Google Cartographer [2] and Graph-Based SLAM [13].

A review of 2D laser scan based SLAM methods is presented in [14]. Five algorithms were evaluated: as the examples of single hypotheses tracking there were HectorSLAM and tinySLAM, multiple hypotheses SLAM was presented by GMapping, and graph-based SLAM – by KartoSLAM and LagoSLAM. The accuracy of the SLAM methods was evaluated by a normalized distance between the occupied cells of an output and the ground truth maps. The original tinySLAM is claimed to have the greatest error in comparison with the others while GMapping shows the best accuracy. However, these results were obtained on a simulated ($12.2\ m \times 11.7\ m$) and a small real world ($4.6\ m \times 4\ m$) indoor environments.

The aim of this work is to compare three SLAM algorithms: improved tinySLAM [15], original GMapping and new Google Cartographer. They were chosen as the implementations of the ideas described above. In addition, Google Cartographer's limits of application are estimated using the dataset not mentioned in the paper it introduced by.

The rest of the paper is organized as follows: Section II. gives a brief description of the tested SLAM algorithms. Their comparison is provided in Section III. Finally, Section IV. concludes the paper.

II. Description of algorithms

A. tinySLAM

As it was mentioned above tinySLAM tracks a single hypothesis about the actual world state. The method is based on a straightforward prediction-correction loop, so it has a compact and simple implementation.

The map is stored as a grid of cells. Each cell keeps the probability of being occupied for the part of an environment it corresponds to. The probability is updated by the following formula:

$$map_{n+1}(x, y) = (1 - q) \cdot map_n(x, y) + q \cdot value \quad (1)$$

where:

- $map_n(x, y)$ – previously estimated probability value;
- $value$ – an observed cell occupancy: 1 if a laser beam stops inside the cell and 0 otherwise.
- q – a quality measure of the observed *value* ($0 \leq q \leq 1$);

The pose is refined with a Monte-Carlo-based scan matcher. It tries to add normally distributed (zero mean, customizable dispersion) values to each component of the pose to find a new pose that leads to a better correspondence between the map and the last obtained scan. The correspondence is estimated as a sum of occupancies of cells that contain laser scan points.

Several heuristics are implemented by tinySLAM since the scan matcher is not robust enough. Blurring of occupied map parts increases the probability of the cells in the neighborhood of a scan point. The original implementation also interpolates laser scan data to increase the number of range readings and uses the result for scan matching.

Another heuristic is proposed in [15]. It adds laser scans to the map with a lower "quality" value if the robot pose has been refined by the scan matcher. Modified cell model introduced in the paper returns the average as the final cell's occupancy probability.

B. GMapping

GMapping is based on Rao-Blackwellized particle filter (RBPF) [16] that estimates a posterior $p(x_{1:t}|z_{1:t}, u_{0:t})$ about potential robot trajectories $x_{1:t}$ using observations $z_{1:t}$ and odometry $u_{0:t}$ data. The posterior is approximated with a set of points (particles) with corresponding probabilities (weights). The particle with the maximal weight is treated as the actual world state. The weight of a particle is updated with a correspondence measure between a new scan and the map estimated by a scan matcher.

GMapping enhances Rao-Blackwellized mapping with the following techniques:

- improved proposal function decreases robot pose uncertainty in the prediction step of prediction-correction loop;

- selective resampling deals with the particle depletion problem.

A grid cell model used by GMapping keeps the averages of occupancy and position of all obstacles found by a laser rangefinder in this cell.

GMapping uses a kind of a gradient descent method to match scans. On each iteration several predefined directions are tested. The position along a direction that has the maximal matching score is chosen as the initial position for the next iteration. The scan score is computed as follows:

$$score(scan, map) = \sum_{p \in scan} e^{-\frac{1}{\sigma} \cdot d(p,map)^2} \quad (2)$$

where:

- p – a scan point;
- $d(p, map)$ – the minimal distance between p and an obstacle stored in the map (one obstacle point per cell);
- σ – the predefined dispersion.

GMapping typically requires less than 80 particles even in large enough environments ($250\ m \times 250\ m$) to build accurate maps (30 particles are used by default) [3].

C. Cartographer

Google Cartographer stores a map of an environment as a graph where every vertex presents a submap and scans got after the corresponding submap has been created. The edges represent transformations between corresponding submaps. So an optimization step to make the map consistent appears besides a scan matching process.

Every cell of a Cartographer submap has a probability to be occupied. It is updated in case the submap has not been fully constructed yet. The following formula presents the rule of updating the cell value being hit (or miss):

$$M_{new}(cell) = \mathrm{odds}^{-1}(\mathrm{odds}(M_{old}(cell) \cdot \mathrm{odds}(p_{hit})))$$

where:

- $M_{old}(\cdot)$ – the old probability of the cell;
- p_{hit} – the probability of the cell being hit;
- $\mathrm{odds}(p) = \frac{p}{1-p}$.

The main idea of the approach in the scan matching is to minimize the cost functional. Thus the scan matching process goes through a minimization of the following functional:

$$\arg\min_{\xi} \sum_{k=1}^{K} (1 - M_{smooth}(T_\xi h_k))^2 \quad (3)$$

where:

- $M_{smooth}(x)$ – the value of a cell x smoothed by values in its neighborhood;
- h_k – a cell involves a point from a laser scan;
- T_ξ – the transformation matrix that shifts a point h_k by ξ;
- ξ – the offset vector $(\xi_x, \xi_y, \xi_\theta)^T$.

This minimization is presented as a brute-force approach with a branch-and-bound modification. To make this process work fast the Ceres [17] library is used.

The optimization problem is solved by a brute-force approach too. It is required to search not only one value ξ as it was searched in (3) but a set of all suitable transformations between vertices. The Ceres library is used to provide these optimizations too. These optimizations take a lot of time so this work is done every few seconds parallelly and correct the results that are already logged. It makes Cartographer not real-time in the strong terms.

III. Evaluation

The accuracy of a SLAM algorithm can be estimated by comparison of an output trajectory with the ground truth one. This comparison should be performed on the datasets collected in real world environments in order to evaluate applicability of the SLAM method.

The MIT Stata Center dataset [18] is a one of a few laser scan datasets that provides ground truth trajectories. They can be extracted by running a localization method on the floor plan provided with the dataset. It contains 84 sequences but some of them are multi-floor (unlike Cartographer, both tinySLAM and GMapping are not supposed to work in such environments) and some sequences are not supplied with the corresponding floor plans. So it was selected 11 sequences to test considered SLAM algorithms.

The Willow Garage dataset [19] does not provide ground truth trajectories, so the accuracy of the algorithms cannot be estimated quantitatively. In this case SLAM algorithms can be compared qualitatively by visual matching of output maps. The authors of Cartographer use this dataset to demonstrate performance of the algorithm. So one representative sequence was chosen to test tinySLAM and GMapping. To the best of our knowledge, evaluation results on this dataset are not publicly available for these algorithms.

Both of these datasets were collected using PR2 robot platform equipped with the Hokuyo UTM-30LX laser scanner.

Since all of the algorithms are probabilistic their output may vary from one run to another. So the root-mean-square error (RMSE) between output and ground truth robot trajectories should be computed to estimate accuracy of the SLAM algorithms. This evaluation was performed on the MIT Stata Center dataset. The trajectories were compared with the TUM's SLAM evaluation tool [20]. All algorithms were tested "as is", i.e. none of algorithms' parameters were tweaked for a specific sequence.

The mean and dispersion values of an RMSE over several runs are presented in Table I and Fig. 2 visualizes the values for some sequences. The table contains two columns for Cartographer since it can perform the final optimization of the robot trajectory after processing of an entire data sequence. The "Cartographer (offline)" column contains the RMSE values after the final optimization. The "Cartographer (online)" column shows the RMSE of trajectory data collected at run time. However, Cartographer was unable to optimize trajectories for some data sequences (the "—" mark in Table I). It may be caused by errors in the tested implementation of the algorithm.

TABLE I. RMSE VALUES

Sequence	Length, m	Trajectory RMSE, m			
		GMapping	tinySLAM	Cartographer (online)	Cartographer (offline)
2011-01-19-07-49-38	68	0.216 ± 0.012	1.280 ± 0.640	0.188 ± 0.023	0.191 ± 0.001
2011-01-20-07-18-45	76	0.219 ± 0.012	0.254 ± 0.045	0.219 ± 0.002	0.221 ± 0.001
2011-01-21-09-01-36	87	0.212 ± 0.029	0.242 ± 0.005	0.217 ± 0.003	0.205 ± 0.001
2011-01-24-06-18-27	87	0.290 ± 0.035	0.254 ± 0.006	0.217 ± 0.001	0.217 ± 0.001
2011-01-25-06-29-26	109	0.208 ± 0.008	0.260 ± 0.005	0.232 ± 0.001	0.232 ± 0.002
2011-01-27-07-49-54	94	0.266 ± 0.012	0.620 ± 0.030	0.266 ± 0.004	—
2011-01-28-06-37-23	145	2.388 ± 1.949	2.280 ± 0.750	0.360 ± 0.069	0.354 ± 0.004
2011-03-11-06-48-23	245	0.365 ± 0.208	0.860 ± 0.390	1.152 ± 0.601	1.348 ± 0.001
2011-03-18-06-22-35	80	0.145 ± 0.023	0.103 ± 0.008	0.145 ± 0.021	—
2011-04-06-07-04-17	95	0.190 ± 0.002	0.343 ± 0.025	0.201 ± 0.002	—
2011-10-20-11-38-39	264	0.352 ± 0.003	5.486 ± 2.603	2.217 ± 0.021	—

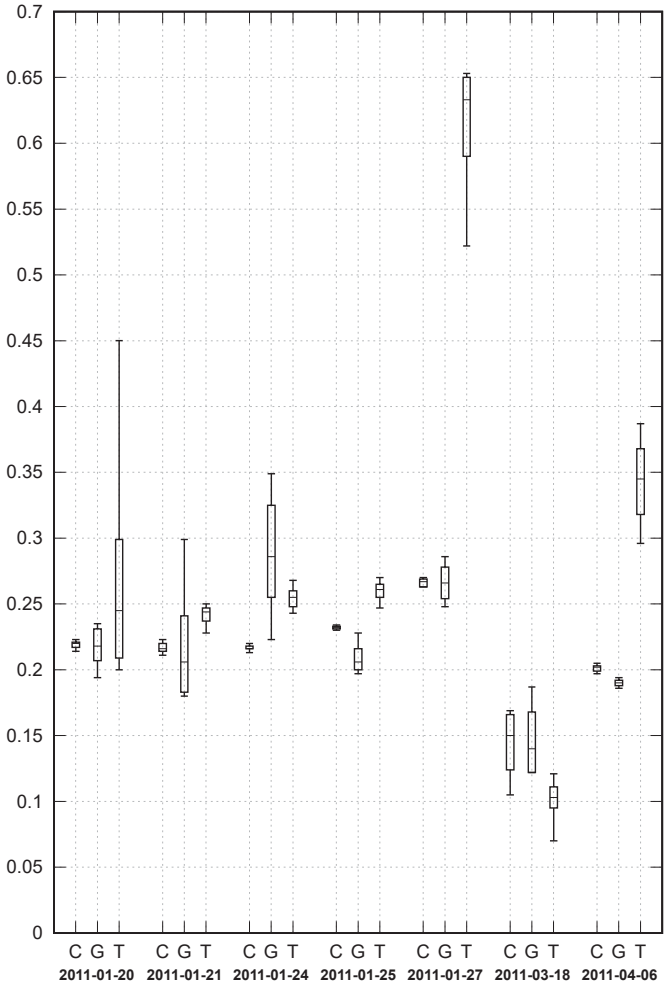
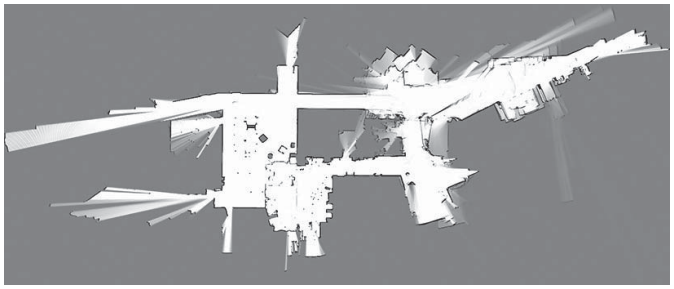
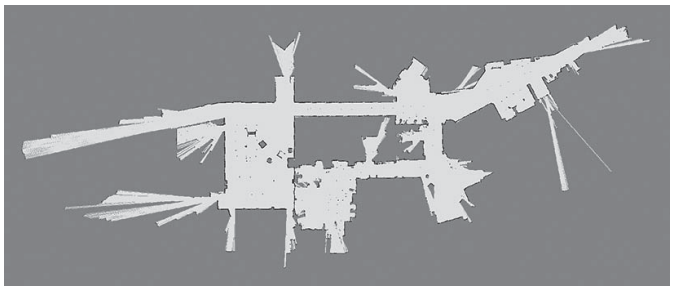


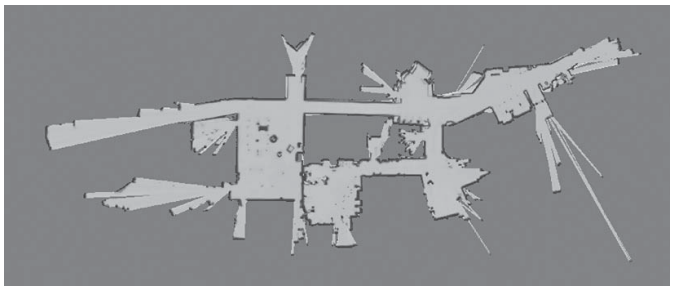
Fig. 2. RMSE values: Cartographer (C), GMapping (G) and tinySLAM (T)



(a) Cartographer



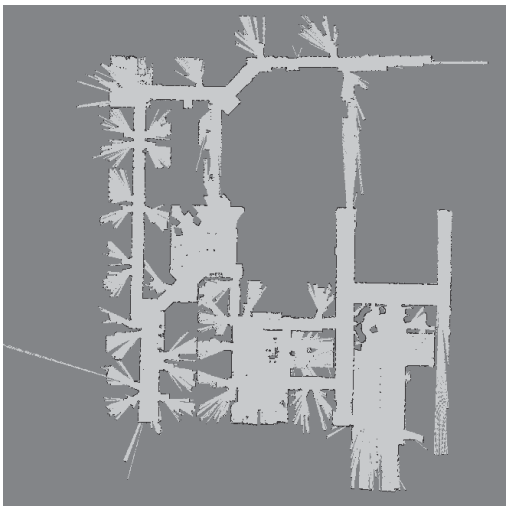
(b) GMapping



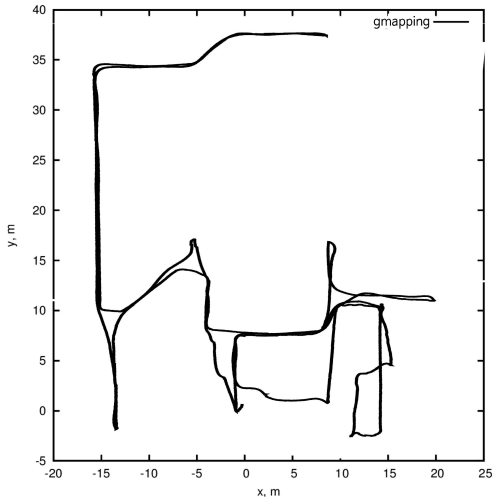
(c) tinySLAM

Fig. 3. Maps built on the "2011-03-11-06-48-23" sequence

The results show that, in general, GMapping and Cartographer have comparable accuracy on the short trajectories (less than 100 m). At the same time, Cartographer has a lower dispersion of the results, i.e. its ouptut are more predictable. However, Cartographer may fail on sequences with long featureless halls, for example, on the "2011-03-11-06-48-23" sequence (Fig. 3a). Multiple hypotheses tracking implemented in GMapping usually produces far more accurate results on

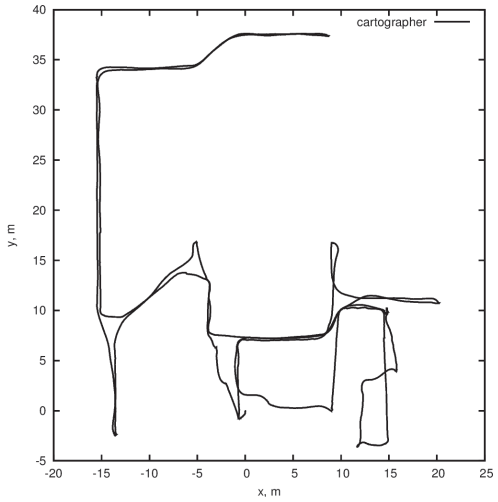


(a) The map built using GMapping



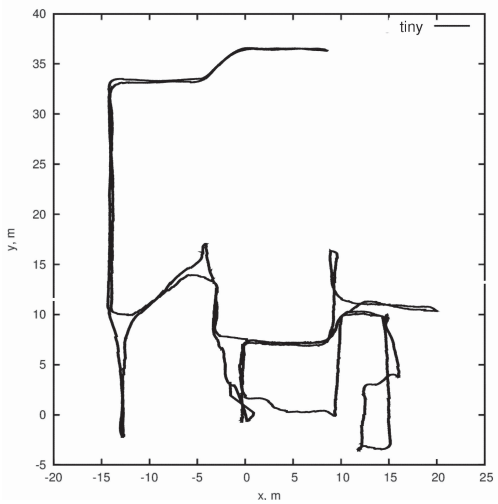
(b) The trajectory built using GMapping

(c) The map built using Cartographer



(d) The trajectory built using Cartographer

(e) The map built using tinySLAM



(f) The trajectory built using tinySLAM

Fig. 4. Maps built with different SLAMs

the input of this kind (Fig. 3b). The tinySLAM method also fails to built a consistent map on most of such sequences but sometimes it has a better output quality than Cartographer (Fig. 3c). A successfull completetion of the final trajectory optimization in the Cartographer algorithm has a small impact on the RMSE value, but it significantly reduces the dispersion which may be useful if the quality of an environment map is more important than the trajectory accuracy.

The quality of maps constructed by considered algorithms was evaluated on the "2011-08-04-23-46-28" sequence from the Willow Garage dataset. The maps and trajectories presented in Fig. 4 are the best ones over several runs of the algorithms. According to the figure all tested algorithms are able to build a consistent enough map, but the map built by tinySLAM (Fig. 4e) has more artifacts than the others. It's worth noting that tinySLAM required the largest number of runs to get the consistent map which means that it is less robust. Nevertheless, all output trajectories are close enough to each other.

IV. Conclusion

Three implementations of popular 2D laser scan based SLAM algorithms were compared: tinySLAM, GMapping and Cartographer. The evaluation was performed on two datasets collected in real world indoor environments. The algorithms were run using default configurations though varying some parameters could improve accuracy, but this contradicts the idea of a universal (arbitrary applicable) SLAM method.

It was shown that tinySLAM provides the greatest root-mean-square error on long data sequences because it tracks only one hypothesis and accumalates errors during evaluation. Due to its stochastic nature, tinySLAM may estimate a world state incorrectly, so this leads to errors in a map that cannot be fixed. As it was expected, GMapping provides a smaller error than tinySLAM as it keeps multiple hypothesis about the world state simultaneously and uses a non-stochastic scan matcher.

Cartographer and GMapping have comparable RMSE values on most input sequences. Cartographer typically has a lower error dispersion value, so it is more robust. But it is less accurate on other sequences, so the tested implementation is not universal.

Acknowledgment

Authors would like to thank JetBrains Research* for provided support and materials for working on this paper.

References

[1] R. Siegwart, I. R. Nourbakhsh, D. Scaramuzza, "Introduction to Autonomous Mobile Robots". Cambridge: The MIT Press, 2011.

[2] W. Hess, D. Kohler, H. Rapp, "Real-time loop closure in 2D LIDAR SLAM", *in Proc. of IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1271-1278, 2016.

[3] G. Grisetti, C. Stachniss, W. Burgard, "Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters", *IEEE Transactions on Robotics*, pp. 34-46, 2007.

*JetBrains Research website: https://research.jetbrains.org

[4] B. Steux, O. E. Hamzaoui, "tinySLAM: A SLAM algorithm in less than 200 lines C-language program", *in Proc. of the 11th International Conference on Control Automation Robotics Vision*, pp. 1975-1979, 2010.

[5] F. Lu, E. Milios, "Robot pose estimation in unknown environments by matching 2D range scans", *in Proc. of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 935-938, 1994.

[6] K. Konolige, K. Chou, "Markov Localization using Correlation", *in Proc. of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1154-1159, 1999.

[7] S. Thrun, W. Burgard, D. Fox, "Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)". Cambridge: The MIT Press, 2005.

[8] N. Zikos, V. Petridis, "L-SLAM: Reduced dimensionality FastSLAM with unknown data association", *in Proc. of IEEE International Conference on Robotics and Automation*, pp. 4074-4079, 2011.

[9] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit, "Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges", *in Proc. of the 18th international joint conference on Artificial intelligence*, pp. 1151-1156, 2003.

[10] A. I. Eliazar, R. Parr, "DP-SLAM 2.0", *in Proc. of the IEEE International Conference on Robotics and Automation*, pp. 1314-1320, 2004.

[11] F. LuE. Milios, "Globally Consistent Range Scan Alignment for Environment Mapping". Kluwer Academic Publishers, 1997.

[12] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, W. Burgard, "G^2o: A general framework for graph optimization", *in Proc. of IEEE International Conference on Robotics and Automation*, pp. 3607-3613, 2011.

[13] G. Grisetti, R. Kümmerle, C. Stachniss, W. Burgard, "A Tutorial on Graph-Based SLAM", *IEEE Intelligent Transportation Systems Magazine*, vol. 2, pp. 31-43, 2011.

[14] J. M. Santos, D. Portugal, R. P. Rocha, "An evaluation of 2D SLAM techniques available in Robot Operating System", *in Proc. of 2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pp. 1-6, 2013.

[15] A. Huletski, D. Kartashov, K. Krinkin, "TinySLAM Improvements for Indoor Navigation" *2016 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, Sept 19-21, 2016.

[16] A. Doucet, N. de Freitas, K. P. Murphy, S. J. Russell, "Rao-Blackwellised Particle Filtering for Dynamic Bayesian Networks", *in Proc. of the 16th Conference on Uncertainty in Artificial Intelligence*, pp. 176-183, 2000.

[17] S. Agarwal, K. Mierle, and Others, Ceres solver Web: http://ceres-solver.org.

[18] M. Fallon, H. Johannsson, M. Kaess, J. Leonard "The MIT Stata Center dataset", *The International Journal of Robotics Research*, vol. 32, no. 14, pp. 1695-1699, 2013.

[19] J. Mason, B. Marthi "An object-based semantic world model for long-term change detection and semantic querying", *in. Proc of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3851-3858, 2012.

[20] Computer Vision Group — Useful tools for the RGB-D benchmark, Web: http://vision.in.tum.de/data/datasets/rgbd-dataset/tools.