An Evaluation of 2D SLAM Techniques Available in Robot Operating System

João Machado Santos, David Portugal and Rui P. Rocha

Abstract— **In this work, a study of several laser-based 2D Simultaneous Localization and Mapping (SLAM) techniques available in Robot Operating System (ROS) is conducted. All the approaches have been evaluated and compared in 2D simulations and real world experiments. In order to draw conclusions on the performance of the tested techniques, the experimental results were collected under the same conditions and a generalized performance metric based on the k-nearest neighbors concept was applied. Moreover, the CPU load of each technique is examined.**

This work provides insight on the weaknesses and strengths of each solution. Such analysis is fundamental to decide which solution to adopt according to the properties of the intended final application.

I. Introduction

Simultaneous Localization and Mapping (SLAM) is one of the most widely researched topics in Robotics. It is useful for building and updating maps within unknown environments, while the robot keeps the information about its location.

Proprioceptive sensors are subject to cumulative errors when estimating the mobile robot's motion, the high dimensionality of the environment that is being mapped, the problem of determining whether sensor measurements taken at different points in time correspond to the same object in the world, and the fact that the world changes over time, represent the biggest challenges in SLAM [1].

The Robot Operating System (ROS) is the most popular robotics framework nowadays. It provides a set of tools, libraries and drivers in order to help develop robot applications with hardware abstraction [2]. ROS enables researchers to quickly and easily perform simulations and real world experiments.

All five SLAM techniques analyzed in this work are available in ROS and have been tested in 2D simulations through Stage and on a custom Arduino-based Robot [3]. The research presented in this article is a first step for our ultimate goal, which is to propose a SLAM technique for Urban Search and Rescue (USAR) scenarios, whose environment often contain smoke and dust particles. Therefore, it is necessary to study the most popular and commonly used approaches and this work will serve as guidance to our later technique, as well as to researchers interested in SLAM and in ROS, in general.

This work has been supported by the CHOPIN research project (PTDC/EEA-CRO/119000/2010), by a PhD grant (SFRH/BD/64426/2009) and by the Institute of Systems and Robotics (project Est-C/EEI/UI0048/2011), all of them funded by the Portuguese science agency "Fundação para a Ciência e a Tecnologia". J.M. Santos, D. Portugal and R.P. Rocha are with the Institute of Systems and Robotics, Univ. of Coimbra, Pólo II, 3030-290 Coimbra, Portugal, email: {jsantos, davidbsp, rprocha}@isr.uc.pt.

978-1-4799-0880-6/13/$31.00 ©2013 IEEE

II. Related Work

Presently, all recognized algorithms for robot mapping have a common feature: they rely in probabilities. The advantage of applying probabilities is the robustness to measurement noise and the ability to formally represent uncertainty in the measurement and estimation process. Most of the probabilistic models used to solve the problem of mapping rely on Bayes rule [1].

Kalman filters (KF) are one of the most popular implementations of Bayes filters [1]. The KF has two distinct phases: Prediction and Update. The prediction phase estimates the state space (prior) from a previous iteration, while in the update phase the estimated state is combined with observations provided by sensors. The result from the update phase is called posterior. Arising from the prior development of the KF, the Extended Kalman Filter (EKF) solves the problem of nonlinearity in the robot pose model. A set of tests on convergence properties and inconsistency issues of the EKF-based solution to the nonlinear 2D SLAM problem is conducted in [4].

Particle filters (PF) are another application of Bayes filters. The posterior probability is represented by a set of weighted particles and each particle is given an importance factor. It assumes that the next state depends only on the current one, *i.e.*, Markov assumption [5]. PFs have the advantage of representing uncertainty through multi-modal distributions and dealing with non-Gaussian noise. Montemerlo *et al.* [6] proposed a new approach called FastSLAM. It makes use of a modified PF to estimate the posterior. Afterwards, each particle possesses K Kalman filters that estimate the K landmark locations. It was shown that the computational effort to execute this algorithm is lower than EFK approaches. Also, the approach deals with large number of landmarks even with small sets of particles and the results remain appropriate. Also, an approach based on PF is proposed in [7]. This work is discussed in more detail in Section III-B.

Equally important are graph-based SLAM algorithms, as they cover some weaknesses of PFs and EKFs techniques [9]. In these SLAM algorithms, the data extracted is used to build a graph. The graph is composed by nodes and arcs. Each arc in the graph represents a constraint between successive poses, which can be a motion event or a measurement event. In order to obtain a map, all the constraints are linearized and a sparse matrix is obtained, representing the sparse graph. This type of algorithms were first presented by Lu and Milios [8]. In their work, pose constraints were retrieved by the scan matching process. However, due to the optimization process

used, the applicability of the algorithm in large scenarios is impracticable. Thrun *et al.* [9] presented the GraphSLAM algorithm, which is based on [8], and evaluated its behavior in large-scale urban environments. This has been possible due to a reduction process, which removes map variables from the optimization process. In addition, Carlone *et al.* [10] also developed a graph-based SLAM approach, which is discussed in Section III-E.

In the last few years, the number of SLAM approaches has increased and the need to compare different approaches grew significantly. Visually inspection of the resulting maps does not allow a correct comparison. So, the need to precisely evaluate the results asks for a more accurate method - a quantitative scale. For instance, in [12], a metric for comparing SLAM algorithms was developed, wherein the result is not evaluated using a reference, but rather by considering the poses of the robot during data acquisition. This fact allows comparison between algorithms with different outputs. Also, the proposed method is independent on the sensor configuration of the mobile robot, but it requires manual editing of the dataset before being applied.

All recognized SLAM evaluation methods rely on standard datasets available to the community. However, these are not compatible with ROS framework yet. Conversely, in this work, a study of the main laser-based 2D SLAM algorithms that are available in ROS is presented. All the tested techniques use occupancy grids as the final output, which are analyzed using a metric for map similarities. The focus is put on the map quality instead of the pose estimation errors, since the mapping output is highly affected by localization issues. The main goal is to provide an overview of the strengths and weaknesses of all five algorithms implemented in ROS and also to provide a simple, yet accurate quantitative comparison, thus defining general guidelines for ROS users to select the algorithm that best fits their requirements.

III. 2D SLAM Algorithms

In this section, a brief description of five SLAM techniques is conducted, namely: HectorSLAM, Gmapping, KartoSLAM, CoreSLAM and LagoSLAM.

A. HectorSLAM

HectorSLAM[1] combines a 2D SLAM system based on robust scan matching and 3D navigation technique using an inertial sensing system [11].

The authors have focused on the estimation of the robot movement in real-time, making use of the high update rate and the low distance measurement noise from modern LIDARs. The odometric information is not used, which gives the possibility to implement this approach in aerial robots like, a Quadrotor UAV or in ground robots operating in uneven terrains. On the other hand, it might have problems when only low rate scans are available and it does not leverage when odometry estimates are fairly accurate. The 2D pose estimation is based on optimization of the alignment of beam endpoints with the map obtained so far. The endpoints are projected in the actual map and the occupancy probabilities are estimated. Scan matching is solved using a Gaussian-Newton equation, which finds the rigid transformation that best fits the laser beams with the map. In addition, a multi-resolution map representation is used, to avoid getting stuck in local minima. Finally, the 3D state estimation for the navigation filter is based on EKF. However, this is only needed when an Inertial Measurement Unit (IMU) is present, such as in the case of aerial robots. Thus, it will not be used in this work.

B. Gmapping

Gmapping[2] is a laser-based SLAM algorithm as described by [7]. Furthermore, it is the most widely used SLAM package in robots worldwide. This algorithm has been proposed by Grisetti *et al.* and is a Rao-Blackwellized PF SLAM approach. The PF family of algorithms usually requires a high number of particles to obtain good results, which increases its computational complexity. Also, the depletion problem[3] associated with the PF resampling process decreases the algorithm accuracy. This happens because the importance weights of particles may become insignificant. Hence, this means that there is a small probability that correct hypothesis can be eliminated.

An adaptive resampling technique has been developed in [7], which minimizes the particle depletion problem, since this process is only performed when is needed. The authors also proposed a way to compute an accurate distribution by taking into account not only the movement of the robotic platform, but also the most recent observations. This decreases the uncertainty about the robot's pose in the prediction step of the PF. As a consequence, the number of particles required is decreased since the uncertainty is lower, due to the scan matching process. In our experiments, the number of particles used by Gmapping was 30.

C. KartoSLAM

KartoSLAM[4] is a graph-based SLAM approach developed by SRI International's Karto Robotics, which has been extended for ROS by using a highly-optimized and non-iterative Cholesky matrix decomposition for sparse linear systems as its solver [13]. A graph-based SLAM algorithm represents the map by means of graphs. In this case, each node represents a pose of the robot along its trajectory and a set of sensor measurements. These are connected by arcs which represent the motion between successive poses. For each new node, the map is computed by finding the spatial configuration of the nodes which are consistent with constraints from the arcs. In the KartoSLAM version available for ROS, the Sparse Pose Adjustment (SPA) is responsible for both scan matching and loop-closure procedures [14]. The higher the number of landmarks, the more

[1] http://www.ros.org/wiki/hector_slam

[2] http://www.ros.org/wiki/gmapping

[3] The particle depletion problem consists in the elimination of a large number of particles from the sample set during the resampling stage.

[4] http://www.ros.org/wiki/karto

amount of memory is required. However, graph-based SLAM algorithms are usually more efficient than other approaches when maintaining a map of a large-scale environments. In the particular case of KartoSLAM, it is extremely efficient, since it only maintains a pose graph.

D. CoreSLAM

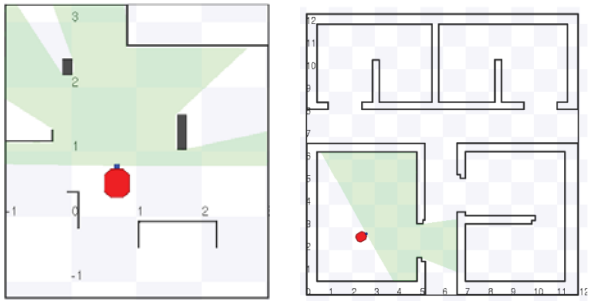
CoreSLAM[5] is a ROS wrapper for the original 200-lines-of-code tinySLAM algorithm, which was created with the purpose of being simple and easy to understand with minimum loss of performance [15]. The algorithm is divided in two different steps: distance calculation and update of the map. In the first step, for each incoming scan, it calculates the distance based on a very simple PF algorithm. The PF matches each scan from the LRF with the map and each particle of the filter represents a possible pose of the robot and has an associated weight, which depends on previous iterations. After the selection of the best hypotheses, the particles with lower weight are eliminated and new particles are generated. In the update step, the lines corresponding to the received scans are drawn in the map. However, of drawing a single point when an obstacle is detected, tinySLAM draws an adjustable set of points surrounding the obstacle.

E. LagoSLAM

The basis of graph-based SLAM algorithms is the minimization of a nonlinear non-convex cost function [10]. More precisely, at each iteration, a local convex approximation of the initial problem is solved in order to update the graph configuration. The process is repeated until a local minimum of the cost function is reached. However, this optimization process is highly dependent on an initial guess to converge. Carlone *et al.* [10] developed a new approach called LagoSLAM[6] (Linear Approximation for Graph Optimization), in which the optimization process requires no initial guess. In addition, the technique can be used with any standard optimizer. In fact, the algorithm available in ROS has the possibility to choose between three different optimizers: Tree-based netwORk Optimizer (TORO)[7], g2o [16] and LAGO [10]. In the experiments conducted, the LAGO optimizer was used. Assuming that the relative position and orientation are independent for each node in the graph, the authors solve a system of equations equivalent to the non-convex cost function. To this end, a set of procedures based on graph theory were presented to obtain a first order approximation of the non-linear system, by means of a linear orientation and a linear position estimation.

IV. Results & Discussion

All five SLAM techniques described were tested using 2D simulations and real world experiments. Simulations were performed in Stage[8], which is a realistic 2D robot simulator integrated in ROS. Additionally, tests were also conducted with a physical robot in a real world scenario, so as to study the behavior of these SLAM packages in the absence of perfect simulated conditions. Despite having perfect conditions in Stage simulations, like noise free odometric and range sensing information, SLAM algorithms assume measurement uncertainty, which may not lead to perfect results. In all experiments, the robot was teleoperated. Note that the abstraction layer provided by ROS allows to use the same code for both simulation and real experiments. HectorSLAM requires a LRF with high update rates. The update rate of the Hokuyo URG-04LX-UG01 LRF used in the experiments is 10 Hz and Stage uses a similar maximum update rate. In order to deal with this, the robot was driven with low angular and linear speed. In the tests that were conducted, the output of each approach, described previously, was the respective generated 2D occupancy grid map.



(a) MRL Arena (4.57×4.04m) (b) 1r5map (12.24×11.67m)

Fig. 1: Maps used in the simulation experiments.

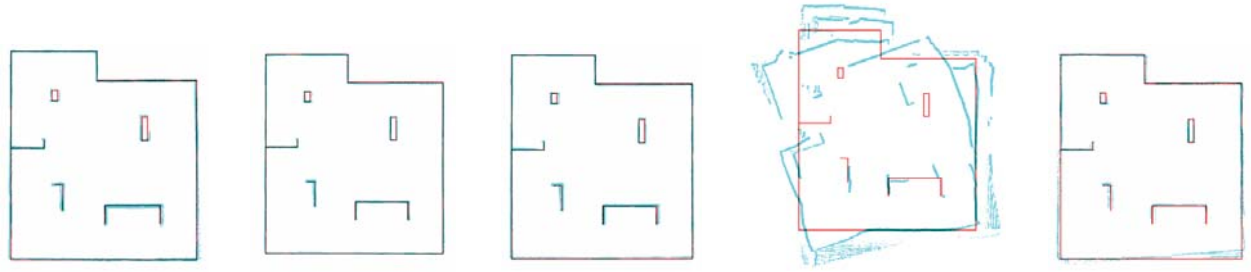
TABLE I: Error estimation for each algorithm in the MRL Arena (Simulation Experiments).

Simulation Experiments				
HectorSLAM	Gmapping	KartoSLAM	CoreSLAM	LagoSLAM
0.4563	0.4200	0.5509	11.8393	1.4646

To evaluate the quality of the maps obtained, an analysis of the error between the generated map and the ground truth was conducted. A performance metric based on the k-nearest neighbor concept was used. To that end, the best fit alignment between the ground truth and the map obtained is computed (see Fig. 2), using intensity-based image registration tools.

The process works as follows: the resulting map of each algorithm is binarized. The binarized map only contains boundaries and obstacles of the scenario. Afterwards, the binarized map is aligned narrowly with the respective ground truth using a set of Matlab functions available in the Image Processing Toolbox. Since both ground truth map and the generated map are aligned, the distance from each occupied cell of the ground truth map to the nearest cell in the resulting map is determined using *knnsearch*, which computes the k-nearest neighbor cells (in this case $k = 1$). The sum of all distances obtained is then divided by the number of occupied cells in the ground truth map. This error metric provides a normalized measure of distance (in terms of cells), which can be applied in any generic occupancy grid, as long as the ground truth map is available.

[5] http://www.ros.org/wiki/coreslam
[6] https://github.com/rrg-polito/rrg-polito-ros-pkg
[7] http://www.openslam.org/toro.html
[8] http://www.ros.org/wiki/stage



(a) HectorSLAM (b) Gmapping (c) KartoSLAM (d) CoreSLAM (e) LagoSLAM

Fig. 2: Maps obtained through simulation in the MRL arena environment. Red represents the ground truth and blue represents the final map.

A. Simulation Tests

Stage simulations were performed using two different maps: the *MRL Arena* and the *1r5map*, which are shown in Fig. 1. Special focus is given to the former since the MRL arena is used in both simulation and real world experiments. The 1r5map enables the analysis of the behavior of the SLAM techniques in a larger scenario and with less features per square foot (*cf.* Fig. 1b). This is particularly important to analyze the dependency on landmarks of each approach.

In the simulation experiments, the model of the range sensor was defined just like the sensor used in real world experiments: the Hokuyo URG-04LX-UG01, which has a maximum range of about 5.6 meters. Teleoperation was executed using the keyboard[9]. All the sensing and actuator data from the robot (LFR, odometry, velocity commands, etc.) was recorded previously and then played back for each algorithm. Thus, all SLAM packages were tested under the same exact conditions. This was only possible due to the *rosbag* tool[10].

For each algorithm, the resolution of the final map was set to 0.01 meters/pixel. In order to mitigate the low scanning rate, the number of sub-maps used in HectorSLAM was defined as 5. Since, each sub-map has half resolution of its precedent sub-map, the scan matching process is more accurate, *i.e.*, the scan matching performance is higher in lower resolution maps. In all experiments, the default parameters were used. For example, as mentioned before, the number of particle for the Gmapping algorithm was 30.

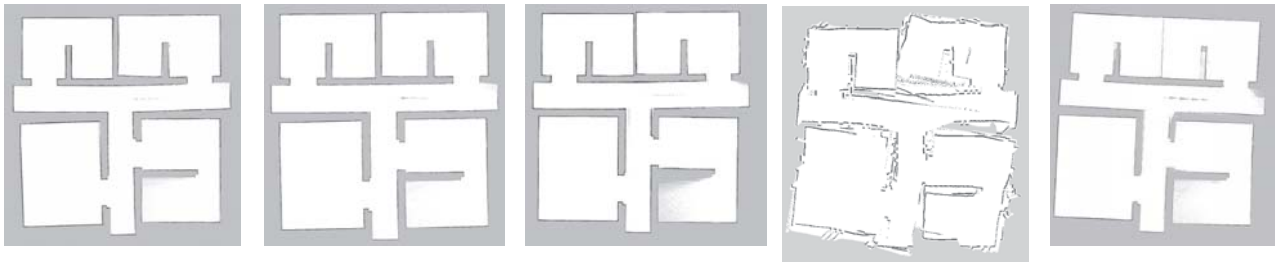
Analyzing the simulations results in the MRL arena, and according to Table I and Fig. 2, Gmapping and HectorSLAM generated the map with lowest and similar error. On the other hand, KartoSLAM presented a slightly greater error, while the results of CoreSLAM presented the highest error value. Gmapping is an extremely optimized PF algorithm with an improved resampling process, and this justifies the quality of the resulting map. Also, the scan matching process of HectorSLAM showed its efficiency. Nevertheless, it must be noted that the low speed commands given to the robot, in order to compensate the rate update from the LFR, have some influence in the results. Since both KartoSLAM and LagoSLAM are graph-based SLAM approaches, comparing the error between them is interesting. Both mapped successfully the arena. However, LagoSLAM obtained the greatest error (excluding CoreSLAM). which can be explained by the impressive performance of the SPA solver method that KartoSLAM employs. Nevertheless, the quality of the resulting map obtained with LagoSLAM is still appropriate.

TABLE II: Error estimation for each algorithm in the 1r5map.

Simulation Experiments				
HectorSLAM	Gmapping	KartoSLAM	CoreSLAM	LagoSLAM
7.4581	5.3670	5.4380	171.5218	9.3041

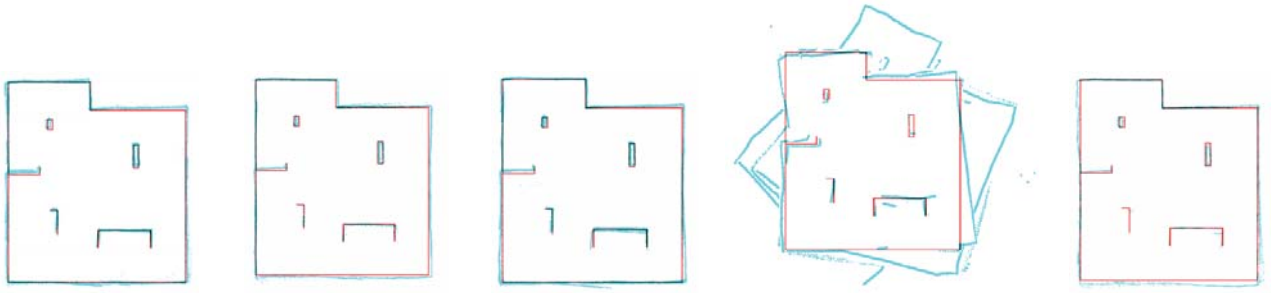
In order to compare all the SLAM approaches in a different scenario, a series of simulations using 1r5map were also conducted. These results are shown in Table II Fig. 3. The 1r5map is a relatively large map with a low number of distinctive landmarks. In this case, HectorSLAM obtained a higher error value than Gmapping. One of the reasons is the fact that HectorSLAM relies largely in scan matching between successive measurements. The full potential of HectorSLAM could not be observed due to the properties of the sensor used in these simulation experiments. Beyond that, due to the reduced number of landmarks, the error grows continuously, since the scan matching process is not fed with enough information. Additionally, since it is not using odometry information, a few issues arise when traversing long corridors with fixed width. As a consequence, the inferior result obtained with HectorSLAM in this test are not surprising. Once again, the Gmapping algorithm presents exceptional results, which reveal the accuracy of PF approaches. KartoSLAM revealed the robustness of graph-based SLAM approaches, since it obtained the second lowest error value. Once again, LagoSLAM obtained an higher error value than KartoSLAM and CoreSLAM was the worst performing algorithm. Since the error values are obtained via the euclidean distance between points in the ground truth and the nearest point in the map, the errors obtained in the 1r5map map are greater than in the other experiments due to the larger dimensions of the map, this is particularly visible in the case of CoreSLAM.

[9] http://www.ros.org/wiki/teleop_twist_keyboard
[10] http://www.ros.org/wiki/rosbag



(a) HectorSLAM (b) Gmapping (c) KartoSLAM (d) CoreSLAM (e) LagoSLAM

Fig. 3: Occupancy Grid Maps obtained through simulation in the 1r5map environment.



(a) HectorSLAM (b) Gmapping (c) KartoSLAM (d) CoreSLAM (e) LagoSLAM

Fig. 4: Performance Analysis in the real world. Red represents the ground truth and blue represents the final map.

TABLE III: Error estimation for each algorithm in the MRL Arena (Real World Experiments).

Real World Experiments				
HectorSLAM	Gmapping	KartoSLAM	CoreSLAM	LagoSLAM
1.1972	2.1716	1.0318	14.75333	3.0264
0.5094	0.6945	0.3742	7.9463	0.8181
1.0656	1.6354	0.9080	7.5824	2.5236

B. Real World Tests

In the real world experiments, three runs with different trajectories and initial positions were performed using a Stingbot[11] robot [3], equipped with an Hokuyo URG-04LX-UG01 and an Asus eeePC 1025C, running Ubuntu 11.10 and ROS Fuerte. Once again, all the data was previously recorded and subsequently played back for each algorithm. Tests were conducted at the real-world MRL arena. The algorithm parameters used in the simulation experiments were again adopted.

Fig. 4 shows that all five techniques were able to map the scenario successfully. The error obtained for each algorithm is shown in Table III. As can be seen, in general all techniques led to worse results than in simulation. This slight performance hit is due to the existence of estimation errors in the robot position and noise on the laser scanning data, while mapping the real world MRL arena. An analysis of the error can give a more accurate information about the performance of the algorithms.

Despite the differences between virtual and real world environments, the results extracted from both setups follow some general trends, in particular for HectorSLAM, Gmapping and LagoSLAM. According to the authors of [15], CoreSLAM can have great performance in several disparate environments; however this claim is not backed up by the results extracted from our experiments.

In the KartoSLAM algorithm, the error obtained in the real world experiments was not much larger than the error in simulations. In fact, generally KartoSLAM was the best performing technique in the real world, being less affected by noise than the other methods. This can be explained, not only due to the performance of the SPA solver used in KartoSLAM, but also because it is a full SLAM approach, *i.e.* the map is obtained using the entire path and map and not only the most recent map and pose. The lower results of CoreSLAM in all experiments showed that its loop closure procedure rarely converges. This is clear in the video that shows a real world experiment and all the detailled results[12].

Beyond the error analysis conducted, an evaluation of the computational load using each technique was carried out. A comparison of the CPU load in a Laptop equipped with an Intel Core i7-3630QM and 8Gb of RAM running each algorithm is presented in Fig. 5 and Table IV.

Looking closely at the results, LagoSLAM presented the highest percentages of CPU usage. Moreover, the values obtained are quite distant from the other four algorithms. This can be explained by the process developed to achieve the minimum cost function for the given graph configuration, as referred in Section III-E. The resources needed by the

[11] `http://www.ros.org/wiki/mrl_robots`

[12] Available at: `http://goo.gl/IMTKmt`

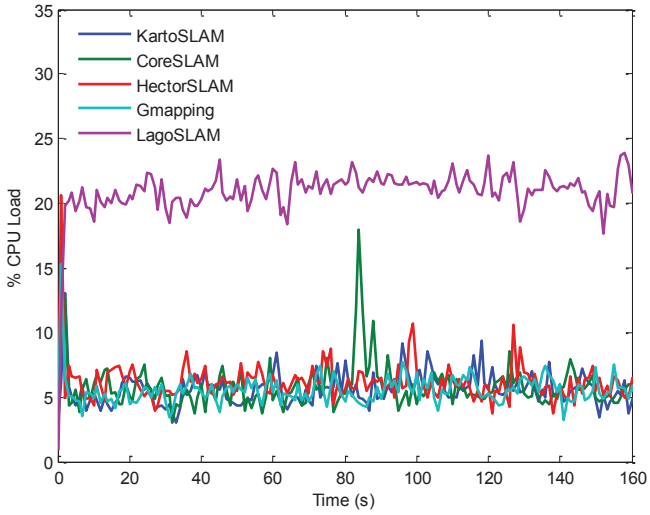


Fig. 5: Evolution of the CPU load of each SLAM method using a real world dataset.

TABLE IV: CPU Load (%) of the 2D SLAM approaches: mean ($\bar{x}$), median ($\tilde{x}$) and standard deviation (σ) values.

	HectorSLAM	Gmapping	KartoSLAM	CoreSLAM	LagoSLAM
$\bar{x}$	6.1107	7.0873	5.4077	5.5213	21.0839
$\tilde{x}$	5.9250	5.5800	5.3000	5.4400	21.2250
σ	1.993	4.4287	1.3018	1.6311	2.1684

other four approaches during the experiments are similar, as seen in Table IV. This CPU analysis reveals that all five algorithms analyzed are quite efficient in terms of resources required and can be adopted online, during field experiments, to map generic 2D scenarios.

C. Discussion

According with our experiments, some ideas can be retained. On one hand HectorSLAM relies only in scan matching and it does not make use of odometry, which could be an advantage or disadvantage depending on the robot and the environment's characteristics. On the other hand, ideally it should be tested with specific hardware such as a high rate LFR.

Gmapping showed its robustness in all experiments, since in every experiment the error and CPU load always remained low. It combines both scan matching and odometry in order to minimize the number of particles.

Both KartoSLAM and LagoSLAM are graph-based SLAM approaches, but their results were distinctively different. KartoSLAM provided accurate maps with lower CPU load, while LagoSLAM generated maps with higher error and CPU load. The reasons behind such discrepancies are related with the distinct processes of graph configuration and graph optimization of the two techniques.

Lastly, CoreSLAM achieved the less impressive results and it is possible to denote a lack of convergence in its loop closure mechanism. CoreSLAM uses a simple PF which requires more particles, but has a lower computation power associated to each particle. According to [15], CoreSLAM uses a very simple PF to match LFR readings with the map, which could lead to an erroneous position estimation. Additionally, the focus of the original work was to provide a simple SLAM technique with the ability to navigate within long corridors without losing its location, and not the loop closing system.

V. Conclusions

In this work, five representative 2D SLAM algorithms available in ROS were tested through simulations and in real world experiments. A discussion of the weaknesses and strengths of each solution has been done. An accurate overview of each of the 2D SLAM techniques available for ROS was provided to shed light on the choice of an approach according to one's requirements.

In future work, we intend to develop a 2D SLAM technique in ROS for low visibility indoor scenarios, *e.g.*, due to smoke. This new technique will possibly adapt Gmapping or KartoSLAM, due to the observed performance in this article, and extend them with more sensing input information beyond LRFs; *e.g.*, sonars, IMUs and/or a dust sensor.

References

[1] S. Thrun., W. Burgard, D. Fox., *Probabilistic Robotics*, MIT Press, 2005.
[2] M. Quigley et al., ROS: an open-source Robot Operating System. *In IEEE International Conference on Robotics and Automation(ICRA)*, Workshop on Open Source Software, 2009.
[3] A. Araújo, D. Portugal, M. Couceiro and R. P. Rocha. Integrating Arduino-based Educational Mobile Robots in ROS, In *Int. Conf. on Autonomous Robot Systems*, Lisbon, Portugal, April 25-29, 2013.
[4] S. Huang, G. Dissanayake. Convergence and Consistency Analysis for Extended Kalman Filter Based SLAM, In *IEEE Trans. on Robotics*, 2(5), Oct. 2007.
[5] S. Thrun, D. Fox, W. Bugard, F. Dellaert, Robust Monte Carlo Localization for Mobile Robots, In *Artificial Inteligence*, 128, 2001.
[6] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit, FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem, In *AAAI National Conference on Artificial Intelligence*, 2002.
[7] G. Grisetti, C. Stachniss, W. Burgard. Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters, In *Trans. on Robotics* , 23(1), Feb. 2007.
[8] F. Lu, E. Milios, Globally Consistent Range Scan Alignment for Environment Mapping, In *Autonomous Robots*, 1997.
[9] S. Thrun, M. Montemerlo. The GraphSLAM Algorithm With Applications to Large-Scale Mapping of Urban Structures, In Proc. of the *Int. Journal on Robotics Research*, 2005.
[10] L. Carlone, R. Aragues, J.A. Castellanos, and B. Bona. A linear approximation for graph-based simultaneous localization and mapping, In Proc. of the *Int. Conf. Robotics: Science and Systems*, 2011.
[11] S. Kohlbrecher, J. Meyer, O. Von Stryk, U. Klingauf. A Flexible and Scalable SLAM System with Full 3D Motion Estimation, In the *Int. Symp. on Safety, Security and Rescue Robotics (SSRR)*, Nov. 2011.
[12] R. Kümmerle, B. Steder, C. Dornhege, M. Ruhnke, G. Grisetti, C. Stachniss, A. Kleiner. On Measuring the Accuracy of SLAM Algorithms, *Autonomous Robots*, 27(4), Nov. 2009.
[13] R. Vincent, B. Limketkai, M. Eriksen. Comparison of indoor robot localization techniques in the absence of GPS, In Proc. of *SPIE*: Detection and Sensing of Mines, Explosive Objects, and Obscured Targets XV of Defense, Security, and Sensing Symposium, April 2010.
[14] K. Konolige, G. Grisetti, R. Kümmerle, B. Limketkai, R. Vincent, Efficient Sparse Pose Adjustment for 2D Mapping, In Proc. of *Int. Conf. on Intelligent Robots and Systems (IROS)*, Oct. 2010.
[15] B. Steux, O. El Hamzaoui. tinySLAM: A SLAM algorithm in less than 200 lines C-language program, In Proc. of the *Int. Conf. on Control Automation Robotics & Vision (ICARCV)*, Dec. 2010.
[16] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, W. Burgard: g2o: A General Framework for Graph Optimization, *IEEE International Conference on Robotics and Automation* (ICRA), 2011.