A Quantitative Study of Tuning ROS Gmapping Parameters and Their Effect on Performing Indoor 2D SLAM

Yassin Abdelrasoul, Abu Bakar Sayuti HM Saman, Patrick Sebastian
Electrical and Electronic Engineering department
Universiti Teknologi PETRONAS
Bandar Seri Iskandar 32610, Malaysian
yassin.abdohassan@gmail.com,{sayutis,patrick_sebastian}@utp.edu.my

***Abstract*— Simultaneous localization and mapping (SLAM) complexity reduction is a fast progressing research area. Its attraction is owed to the potential commercial benefits of developing low cost yet highly effective SLAM based robotic applications. ROS gmapping package offers a lightweight incorporation of FastSLAM 2.0. The package has been used with different ROS supported robotic platforms and showed remarkable success. However, the effect of the package mapping parameters seem not to be fully exploited, especially with low cost robotic platform with no full ROS support such as Hercules platform. This paper presents a full implementation and performance quantitative evaluation on the gmapping package running on both standard PC and Raspberry Pi processors. We study the effects of tuning the number of particles, the displacement update and the resampling threshold by separately varying each of these parameters to several incremental values and running the algorithm on a recorded dataset. For each run, a grid map was constructed and the performance was evaluated based on mapping accuracy, CPU load and memory consumption. We are then able to propose a tuning guidelines to enlighten the gmapping execution while maintaining high performance.**

Keywords—ROS gmapping tuning; FastSLAM 2.0; Indoor 2D SLAM; Low cost SLAM; Laser range finder

I. INTRODUCTION

One of the most basic yet important features of intelligent mobile robots is the ability to navigate autonomously [1]. Autonomous robots are capable of safely exploring their surroundings without colliding with people or slamming into objects. Simultaneous localization and mapping SLAM enables the robot to achieve such autonomy by answering the questions of how its world looks like (mapping) and also where it stands within that world (localization) [2], [3].

Nowadays the applications arena of intelligent robots is flourishing so fast, thanks to the advancements in sensors and processors design on one hand, and SLAM and path planning algorithms on the other hand. The commonly spreading vacuum cleaners, unmanned aerial vehicles and driverless cars are few examples of what theses robots can do. Perhaps the most challenging problem that SLAM researchers and intelligent robots developers have been facing is simplifying the algorithm so that it runs with less computational effort and consumes reasonable memory to suit mobile robots applications and thus reducing their cost and size to become commercially affordable [2], [4], [5].

Several research efforts are available throughout SLAM literature in attempts to producing lightweight yet accurate algorithms. By going through which, one can conclude that researchers considered eliminating the correlation between landmarks and robot trajectory, adaptively reducing the number of recursive operation, reducing the required memory, using low cost sensors with less frequent measurements and imposing some constraints on the mapping environment.

Following the philosophy of map and robot trajectory correlation elimination, FastSLAM [6], [7] factorizes the SLAM problem by first estimating the robot pose with a set of particles using Rao-Blackwellized particle filter (RBPF). Each particle maintains its own map and since the location of each particle is known, the mapping is then carried out as “mapping with known poses” which results in constructing more accurate maps [8].

To improve the performance of the generally accepted RBPF, the authors of [9] developed a new algorithm (FastSLAM 2.0) that utilizes scan matching technique to incorporate both motion model update and the most recent observations to generate more accurate proposal distribution which results in reduced number of particles required during the sampling step. On top of that, they introduced adaptive resampling technique to decrease the frequency of performing the resampling step and thus maintain a quite “diverse set of particles” and consequently solve the particles depletion problem. This approach has achieved remarkable success and indeed was able to reduce the memory consumption while producing highly accurate grid maps.

Based on the fact that the main structure of most indoor environments is parallel or perpendicular planes, the authors of [1], [7] have introduced an EKF based orthogonal SLAM (OrthoSLAM) algorithm which reduces the computational cost of the algorithm by extracting only orthogonal line segments from the environment and thus reducing the total memory consumption. Despite of EKF inherited data association problem and its relatively limited robustness to environments with large amount of landmarks and nonlinear processes [8], their approach produced quite satisfactory results.

Inspired by the success of OrthoSLAM approach briefly discussed above, the authors of [2] developed RBPF based variant of orthogonal SLAM and were able to reconstruct accurate maps by extracting only orthogonal line segments. It's worth mentioning that they introduced a dynamic reference line direction selection technique that performs automatic robot initial alignment compared to the original OrthoSLAM algorithm in which the robot has to initially get aligned manually with the direction of the majority of the orthogonal planes contained in the environment.

II. RBPF Based SLAM

SLAM in its abstract concept is a problem of estimating the map m and the robot trajectory $x_{1:t}$ given the motion odometric data $u_{1:t-1}$ and the exterioceptive observations $z_{1:t}$ [11], [12]. Clearly, the robot trajectory is needed to estimate the map and vice versa. This makes SLAM a hard problem [13]. RBPF overcomes this issue by factorizing the posterior as below:

$$p(x_{1:t}, m \mid z_{1:t}, u_{1:t-1}) = p(m \mid z_{1:t}, u_{1:t-1}) \cdot p(x_{1:t} \mid z_{1:t}, u_{1:t-1}) \quad (1)$$

Eq. 1 illustrates that, RBPF first estimates the robot pose by evaluating $p(x_{1:t} \mid z_{1:t}, u_{1:t-1})$ and then comes the mapping by evaluating $p(m \mid z_{1:t}, u_{1:t-1})$. Following this strategy, each possible pose "particle" carries its own map for which the pose is known [9] .

As discussed in Section I, FastSLAM 2.0 adapts the same concept of RBPF discussed above and applies it to grid mapping with some modifications regarding the proposal distribution estimation and when to perform resampling [9]. The steps of the algorithm are as follows:

1) *Sampling:* The new set of particles is drawn by first giving an initial estimation of the robot pose using the motion model (odometry). It's then improved by incorporating the most recent observations to evaluate how good the new particles fit with the map obtained so far (scan-matching). If the scan-matcher reports a success, a new proposal is sampled around the pose returned by the scan-matcher. And afterwards, a Gaussian approximation $N(\mu_t^{(i)}, \Sigma_t^{(i)})$ is obtained for the "improved proposal distribution" [9] from which the new poses (particles) $x_t^{(i)}$ are estimated. In the situation of scan-matcher reporting a failure, the initial proposal distribution is used to estimate the new poses (particles).
2) *Importance weighing:* a measure $w_t^{(i)}$ of how well a particle i drawn from the improved proposal distribution represents the target distribution is assigned to each particle based on the importance sampling principal:
$$w_t^{(i)} = \frac{p(x_{1:t}^{(i)} \mid z_{1:t}, u_{1:t-1})}{\pi(x_{1:t}^{(i)} \mid z_{1:t}, u_{1:t-1})}. \quad (2)$$
3) *Adaptive resampling:* depending on the outcome of N_{eff}, which describes how well the target distribution is represented by the particles set obtained from the sampling step, a resampling step might be performed. N_{eff} is computed using Eq. 3.
$$N_{\text{eff}} = \frac{1}{\sum_{i=1}^{N} \left(\tilde{w}^{(i)}\right)^2} \quad (3)$$
4) *Map estimation:* Lastly, the map carried by each particle is updated by taking into account the history of robot poses and the observations obtained so far.

III. ROS Gmapping

The robot operating system (ROS) is a fast growing intelligent robotic applications development framework with support for the vast majority of sensors and efficient implementation for different SLAM, path planning and image processing algorithms. As of the date of writing this paper, ROS offers several SLAM algorithms among which are:

- HectorSLAM
- Gmapping
- KartoSLAM
- CoreSLAM
- LagoSLAM

According to the findings of [14], KartoSLAM, HectorSLAM and Gmapping produce the most accurate maps. These algorithms despite having quite similar performance from map accuracy point of view, are actually conceptually different. That's, HectorSLAM is EKF based, while Gmapping and KartoSLAM are based on RBPF occupancy grid mapping and graph based mapping respectively.

Based on the promising results presented in [9], gmapping can potentially perform well on a limited processing power robotic system such as ours. The algorithm uses a relatively small number of particles to represent the SLAM posterior and reduces the computational effort required to perform resampling to successfully build very accurate maps. Keeping in mind our objectives to run SLAM on a low cost robot with limited processing power and memory capacity as raspberry Pi; we decided to implement and evaluate the gmapping algorithm. For detailed explanation on the steps of gmapping algorithm and the adaptive resampling concept, the reader is strongly advised to refer to the work of [9].

The gmapping offers a flexible way to optimizing the mapping process to fit the application specific needs by tuning some mapping parameters such as number of particles used by

RBPF, the displacement step to process new scan and the resampling threshold.

IV. System Implementation

In this section, we first present our system hardware specifications, then a brief overview of our ROS integration architecture is provided along with the description and implementation details of each involved ROS node.

A. Hardware Specifications

The system is implemented on two Linux machines connected over Wi-Fi (distributed system). Specifically, A PC (core-i5, 2.53 GHz CPU, 6.00 GB RAM) running Ubuntu is utilized for robot teleoperation and mapping visualization on rviz. Then, Hercules 4WD mobile platform equipped with Hokuyo URG-04LX-UG01 laser range finder and controlled by raspberry Pi 2 model B make our low cost mobile robot.

B. ROS Architecture

Fig. 1 illustrates the overall ROS integration architecture of our system. ROS has been configured to run on two machines by following the configuration steps available on [16]. The basic system operation steps are as illustrated below:

- *Issuing motion commands:* using keyboard arrow keys, a forward, turn left or right motion can be performed. Our **motion_controller** node publishes the respective left and right wheels velocities under the topic **/motion.**
- *Motors actuation:* in response to the received motion command, the left and right motors of the robot are actuated. We followed the approach presented on [17], [18] to setup **rosserial_python** package and utilize it to configure our Hercules platform Arduino controller as a remote node subscribed to the topic **/motion** and publishes the estimated odometric pose to the topic **/odometry.**
- *Odometry estimation:* by reading the wheels encoders and using the differential drive model, an estimation of the robot 2D pose is obtained. Our node **tf_frames** receives the odometry data and publishes it to the moving frame **estimated_pose.** The node is developed based on the procedures highlighted on [19], to publish its frames to the **/tf** topic.
- *LRF scan:* after the robot translates or rotates by the displacement specified as per linearUpdate and angularUpdate gmapping parameters respectively, a new laser scan which is published to the topic **/scan** by **hokuyo_node** is processed by **gmapping** node.
- *SLAM:* the gmapping node obtains the odometry pose estimation from **/tf** topic, and the latest laser scan and utilizes them in correcting the robot pose and updating the map. The corrected pose is published to the moving frame **corrected_pose**.
- *Visualization:* rviz is configured to display the built map, both the estimated and corrected robot pose as well as the latest laser scan beams.

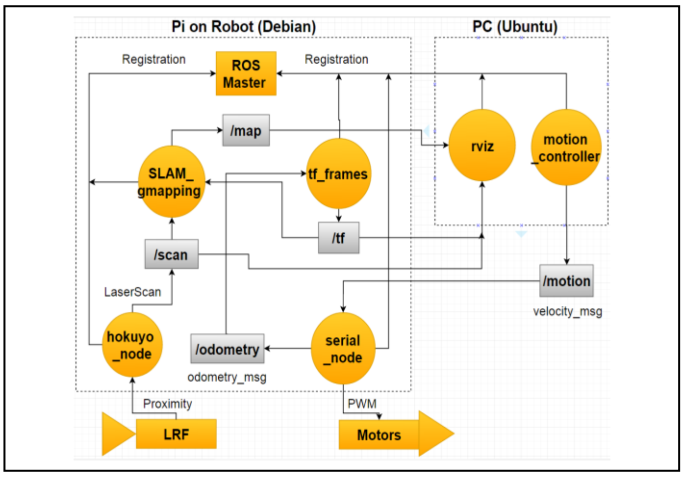

Fig. 1. Our distributed ROS architecture

C. Robot Odometric Pose Estimation Model

In 2D environments, the position (pose) of the robot is represented by a 3D vector containing the robot location and orientation relative to its world. The location is given by the robot **x** and **y** coordinates and the orientation by its yaw angle ***θ***. Our robot having a differential structure, can be modeled using the model presented in [20]. By integrating the left and right wheels displacements **Δsl, Δsr** traveled since receiving the latest motion command, the new robot pose can be estimated using the formula of Eq. 4 to 6 below:

$$Pose = \begin{vmatrix} x \\ y \\ \theta \end{vmatrix} + \begin{vmatrix} \Delta s \cos\left(\theta + \frac{\Delta\theta}{2}\right) \\ \Delta s \sin\left(\theta + \frac{\Delta\theta}{2}\right) \\ \Delta\theta \end{vmatrix} \qquad (4)$$

$$\Delta \mathbf{s} = \frac{\mathbf{\Delta sr + \Delta sl}}{\mathbf{2}} \qquad (5)$$

$$\Delta \boldsymbol{\theta} = \frac{\mathbf{\Delta sr - \Delta sl}}{\boldsymbol{b}} \qquad (6)$$

Eq. 6 implies that the maximum angular velocity a certain robot can perform in one time step is inversely proportional to the robot base width or the distance between its differential wheels **b.**

D. System Coordinate Frames Transformations

Coordinate frames are a common concept in mobile robots equipped with several sensors or legged robots that have several joints. In localization applications, the robot controller does not only need to have all the measurements with respect to the robot center but also needs to continuously know the position of the robot center with respect to the world. With that in mind, it's a necessity in any such robotic application to establish concrete techniques to link between the different interacting coordinate frames, a concept known as coordinate frames transformations.

Frames transformations are basically a set of rotation and translation matrices specifying the linear and rotational offset between each interacting coordinate frames. Our robot has three coordinate frames with the following details:

1) *Fixed frame (map):* A top level frame (parent frame) representing the world map to which the robot pose is related. The frame is static with its origin at (x = 0, y = 0, Yaw = 0).
2) *Robot frame (estimated_pose):* A child frame of the **map** frame attached to the robot center and to which all laser scans are related. The frame origin is located at the most recent pose of the robot estimated by the odometry model. Intuitively, a transform has to be established to continuously relate the robot pose to the map frame. The transformation is made up of the robot pose published by our **serial_node** as translation matrix and its orientation (yaw) as rotation matrix. Since these matrices have different values each time the robot moves; this frame is considered a moving frame.
3) *Gmapping odom frame (corrected_pose):* Another child frame of **map** to which the gmapping localization output is published. Basically, the origin of this frame represents the most probable robot pose estimated by the SLAM algorithms. In the cases when the robot odometric system and the SLAM algorithm are both highly accurate, this frame and the **estimated_pose** frame origins become close to each other.

V. EXPERIMENTAL SETUP

To analyze the performance of gmapping, we've run experiments on a building corridor mapping environment where the robot was tele-operated to explore and record several ROS bags of it, the gmapping was then launched to build maps from the recorded bags with different parameters values by tuning the parameters separately to analyze their effects on the performance. A quick reference to the steps we used in recording the bags can be found in [21].

Our mapping environment is presented in Fig 2. The robot was operated to explore it in two different ways. First by following straight path with no turns and then by following L-shaped trajectory to observe the effect of fast rotations on the scan matcher performance.

VI. RESULTS ANALYSIS

In this section we present a quantitative analysis on our system performance obtained by tuning some of the gmapping parameters. The analysis focused on tuning the number of particles, resampling threshold and the step size between successive laser scans. For each of these parameters, we tried to pick the most informative data to explore and evaluate the effect of tuning the respective parameter on the algorithm computational effort and memory consumption. Furthermore, the generated maps in each case are also evaluated visually to reflect on the mapping accuracy.

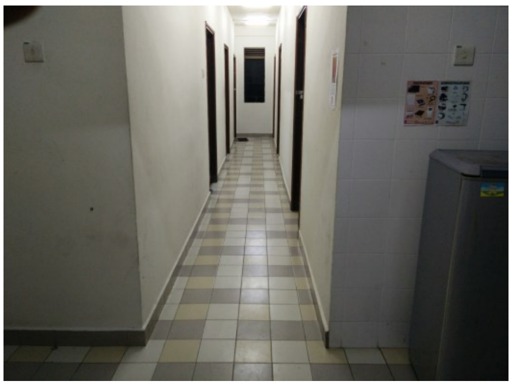

Fig. 2. 1 x 7 m T-shaped building corridor

A. Number of Particles

Although it's known that increasing the particles number contributes to enhancing SLAM accuracy and at the same time introduces more CPU load and memory consumption, it's not intuitively clear how large or small this number should go.

TABLE I. EFFECTS OF INCREASING RBPF NUMBER OF PARTICLES

Particles	CPU Load	Memory (MB)
5	5%	5
15	17%	54
30	17%	54
50	22%	56

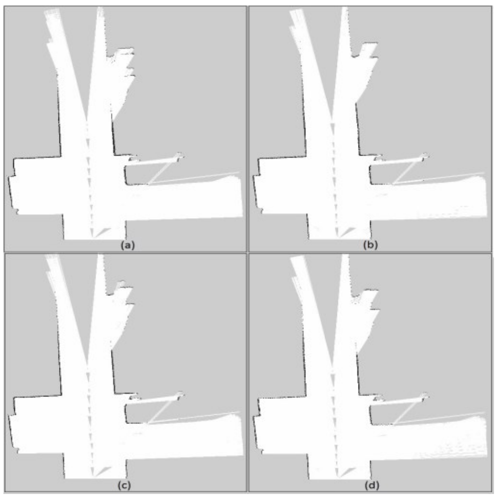

Fig. 3. Maps generated for our T-shaped corridor using (a) 5 particles, (b) 5 particles, (c) 30 particles and (d) 50 particles. These maps were generated using the same data set.

By analyzing the results of Table I, one can see that the CPU load dropped to only 5% in case of using 5 particles compared to 22% when using 50 particles. Moreover, the memory consumption also shows a similar behavior. Interesting enough, the obtained maps seem to have a similar accuracy which proves that gmapping scan matching technique effectively reduces the required number of particles while maintaining the map accuracy provided that the resampling threshold is set properly. Of course, we must keep in mind that the number of particles required to produce satisfactory results relies on the geometry and size of the environment being mapped.

B. Linear Update Step

Gmapping allows adjusting the frequency of processing new observations by the linear and angular update parameters. Increasing the values of these parameters implies that the robot has to translate or rotate a larger distance in order for the latest scan data to be processed and vice versa. On one hand, setting these parameters to large values reduces the CPU load incurred when processing a new scan data. On the other hand, such setting might result in poor constructed maps as the algorithm would miss to capture important environmental features. In the worst cases, the algorithm becomes highly uncertain about the correct pose of the robot, a situation that manifests itself by large pose jumps observed even when the robot is static. The performance results are listed in Table II below and the generated maps can be visualized in Fig. 4.

TABLE II. FFECTS OF ENLARGING THE LASER SCAN PROCESSING STEP

Step (m)	CPU Load	Memory (MB)
0.2	20%	39
0.4	17%	31
0.7	16%	28

The constructed maps as in Fig. 4 illustrate the need of more frequent scans to correctly detect the corners. This imposes navigation speed limit when mapping environments with many corners. Conversely, a robot mapping a straight environment can safely be operated at higher speed and with less frequent laser scans requirements.

C. Resampling Threshold

The resampling threshold controls when to perform resampling step. It's specified as a threshold of the N_{eff} measure below which a resampling must be performed. Due the strong effect of resampling on the possibility of "particles depletion" [9], we decided to tune the resampling threshold by changing its value and observing the performance first with a relatively large number of particles, and then few particles.

This way we will be able to capture the particles depletion effects and also evaluate the effect of resampling on the algorithm performance.

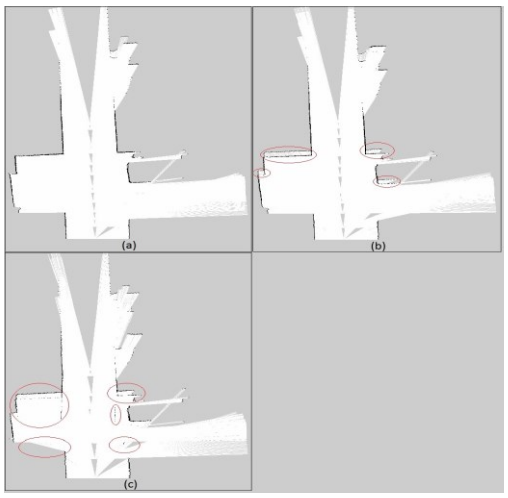

Fig. 4. A typical map inaccuracies observed when increasing the step between laser scans. With 0.4 step size the walls tend to appear duplicated especially at the corners (b). The algorithm misses a corner or erroneously construct a wall when the step is increased to 0.7 (c). In contrast to that, the algorithm correctly aligns the walls and corners when using a small step of 0.1 (a).

TABLE III. EFFECTS OF TUNING THE RESAMPLING THRESHOLD

Resample Threshold	Particles	CPU Load	Memory (MB)
0.1	30	15%	46
0.1	5	12%	42
1	30	23%	58
1	5	22%	54

The results obtained in Table III, prove our expectation regarding the effect of increasing the resampling threshold. That is, with large values the CPU load and memory consumption increase and decrease otherwise. By intuition, one can see the reason behind such behavior; as large resampling threshold results in more frequent resampling carried out and consequently more computational effort and memory consumption.

As visualized in Fig. 5, the effect of particles depletion is observed due to very frequent resampling of the few available particles in case of map (d) and less frequent resampling in case of map (b). That's, the particles available were not enough to represent all the possible robot poses; and thus the mapping became highly erroneous.

From the analysis carried out, one can conclude that to achieve a good performance in terms of mapping accuracy and computational effort, the number of particles can be as low as 5 provided that the resampling threshold is around 0.5 and the step between processed laser scans is sufficiently small. Nevertheless, these values might slightly vary with the size and geometry of the mapping environment.

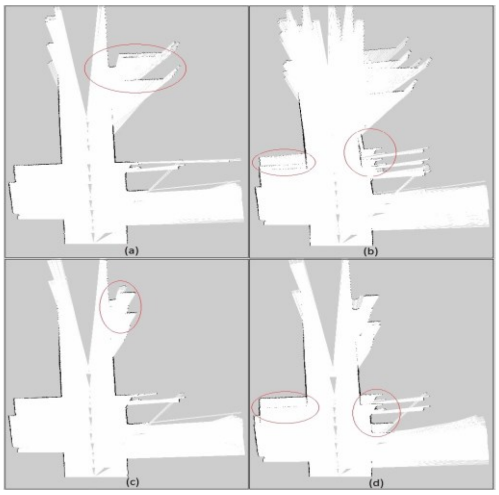

Fig. 5. The constructed maps are highly erroneous with very few particles in both situations when the resampling threshold is very small (b) or very large (d). The resampling threshold lost criticality when the number of particles is large enough (a) and (c).

VII. CONCLUSION

In this paper, we presented a successful implementation of ROS gmapping on low cost Hercules robotic platform using LRF. We also introduced a quantitative guide to tuning some of the algorithm parameters to achieve desired performance.

We started our discussion by reviewing some research efforts in addressing the problem of reducing the complexity of SLAM. Then, we focused on the gmapping algorithm by describing its steps. After that we briefly described our hardware specifications and our system ROS integration architecture along with some implementation details.

From the analysis carried out, one can conclude that to achieve a good performance in terms of mapping accuracy and computational effort, the number of particles can be as low as 5 provided that the resampling threshold is around 0.5 and the step between processed laser scans is sufficiently small. Nevertheless, these values might slightly vary with the size and geometry of the mapping environment.

It's been observed that the speed at which the robot navigates affects the mapping accuracy. In other words, while the robot needs to be operated sufficiently slow to correctly detect corners, it can be operated relatively faster when following straight trajectory. With that in mind, an adaptive speed control technique can be developed to incorporate these results to automatically vary the speed and thus decrease the overall mapping time.

In the case of observing jumping pose or misaligned corners, a good tuning start would be decreasing the linear and angular update values to increase the rate at which the robot observes these features and thus decreasing its pose uncertainty.

REFERENCES

[1] V. Nguyen, A. Harati, and R. Siegwart, "A lightweight SLAM algorithm using Orthogonal planes for indoor mobile robotics," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007, pp. 658–663.

[2] B.-W. Kuo, H.-H. Chang, Y.-C. Chen, and S.-Y. Huang, "A Light-and-Fast SLAM Algorithm for Robots in Indoor Environments Using Line Segment Map," *J. Robot.*, vol. 2011, p. 12, 2011.

[3] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*.

[4] S. Magnenat, V. Longchamp, M. Bonani, P. Rétornaz, P. Germano, H. Bleuler, and F. Mondada, "Affordable SLAM through the co-design of hardware and methodology," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, 2010, pp. 5395–5401.

[5] A. Martinelli, N. Tomatis, and R. Siegwart, "Open challenges in SLAM: an optimal solution based on shift and rotation invariants," in *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, 2004, vol. 2, p. 1327–1332 Vol.2.

[6] M. Montemerlo and S. Thrun, "Simultaneous localization and mapping with unknown data association using FastSLAM," in *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, 2003, vol. 2, pp. 1985–1991 vol.2.

[7] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM: a factored solution to the simultaneous localization and mapping problem," in *Eighteenth national conference on Artificial intelligence*, Edmonton, Alberta, Canada, 2002, pp. 593–598.

[8] A. Monjazeb, J. Z. Sasiadek, and D. Necsulescu, "Autonomous navigation among large number of nearby landmarks using FastSLAM and EKF-SLAM - A comparative study," in *Methods and Models in Automation and Robotics (MMAR), 2011 16th International Conference on*, 2011, pp. 369–374.

[9] G. Grisetti, C. Stachniss, and W. Burgard, "Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters," *IEEE Trans. Robot.*, vol. 23, no. 1, pp. 34–46, Feb. 2007.

[10] V. Nguyen, A. Harati, A. Martinelli, R. Siegwart, and N. Tomatis, "Orthogonal SLAM: a Step toward Lightweight Indoor Autonomous Navigation," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006, pp. 5007–5012.

[11] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part I," *IEEE Robot. Autom. Mag.*, vol. 13, no. 2, pp. 99–110, Jun. 2006.

[12] T. Bailey and H. Durrant-Whyte, "Simultaneous localization and mapping (SLAM): part II," *IEEE Robot. Autom. Mag.*, vol. 13, no. 3, pp. 108–117, Sep. 2006.

[13] W. J. Kuo, S. H. Tseng, J. Y. Yu, and L. C. Fu, "A hybrid approach to RBPF based SLAM with grid mapping enhanced by line matching," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009, pp. 1523–1528.

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

[15] S. Schweigert, "gmapping," *ros.wiki*. .

[16] "ROS on multiple machines," *ros.wiki*. .

[17] S. Dohare, "Rosserial arduino IDE setup," *ros.wiki*, 16-Jun-2016. .

[18] AdamStambler, "Rosserial publisher and subscriber," *ros.wiki*, 12-Aug-2011. .

[19] WilliamWoodall, "Setting up your robot using tf," *ros.wiki*, 07-May-2015. .

[20] R. Siegwart and I. R. Nourbakhsh, *Introduction to autonomous mobile robots*. London, England: MIT press.

[21] DavidRyskalczyk, "How to build a map using logged data," *ros.wiki*, 14-Mar-2014. .