Efficient 2D LIDAR-Based Map Updating For Long-Term Operations in Dynamic Environments

Conference Paper · July 2022

DOI: 10.1109/IROS47612.2022.9982047

4 authors, including:

Elisa Stefanini
Università di Pisa
5 PUBLICATIONS **9** CITATIONS
SEE PROFILE

Alessandro Settimi
Università di Pisa
30 PUBLICATIONS **695** CITATIONS
SEE PROFILE

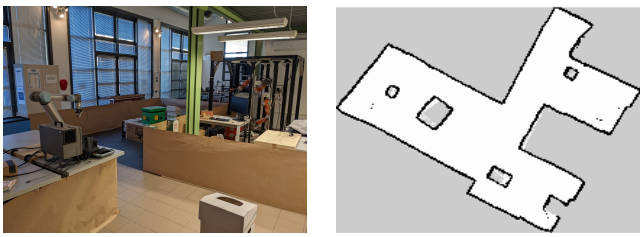
Efficient 2D LIDAR-Based Map Updating For Long-Term Operations in Dynamic Environments

Elisa Stefanini[1,2], Enrico Ciancolini[3], Alessandro Settimi[3] and Lucia Pallottino[1]

***Abstract*— Long-time operations of autonomous vehicles and mobile robots in logistics and service applications are still a challenge. To avoid a continuous re-mapping, the map can be updated to obtain a consistent representation of the current environment. In this paper, we propose a novel LIDAR-based occupancy grid map updating algorithm for dynamic environments taking into account possible localisation and measurement errors. The proposed approach allows robust long-term operations as it can detect changes in the working area even in presence of moving elements. Results highlighting map quality and localisation performance, both in simulation and experiments, are reported.**

I. INTRODUCTION

Today, companies of all sizes use robots to improve their productivity, and mobile robotics is becoming increasingly important for future developments in the sector. In recent years, we are witnessing a shift from Automated Guided Vehicles to Autonomous Mobile Robots (AMR) in many sectors like industry, logistics, and services [1]. To date, the most common use of this type of robot is for transporting materials within factories (e.g., to and from production, assembly lines, and the warehouse) as this is a time-consuming and non-value-added activity for most companies. The characteristics of this type of robots also make their application possible in other sectors such as the agricultural, medical, personal care, and security. An AMR exploits natural navigation and it is capable of redefining paths and avoiding obstacles. Natural navigation implies Simultaneous Localisation and Mapping (SLAM) techniques where the robot maps the environment, navigates, and locates itself by simply "looking" at this environment, without the need of installing further hardware in the work place. The most commonly used perception sensors used for localisation and mapping in industrial environments are laser scanners [2]. Conventional SLAM is used to create an initial image of an unknown environment, and it is not intended to be used as a system for repeatedly updating the same area of the map. This approach could lead to task failure in complex production environments where changes in the map can occur. Industrial environments are usually characterized by a slow rate of change over time since semi-static objects such as pallets, furniture, workstations could change their position when the same task is performed at another time, be it a day or two months later. During AMRs operations, small changes in the environment are well handled by localisation and local obstacle avoidance algorithms; however, as the robot's static map diverges from the current working area, becoming obsolete, navigation performance degrades [3]. Considering environmental changes due to semi-static objects can hence increase the localisation performance of a robotic system [4] through a life-long mapping approach that detects changes in the environment and updates the map reflecting detected changes. With an updated static map, the robot acquires a priori knowledge of the environment, and its task execution time is reduced since the navigation module doesn't have to keep adjusting the trajectory online. Moreover, the map updating process has to consider long-term operation, and the algorithm has to be computationally light and use limited memory [5].



(a) (b)

Fig. 1: The CrossLab Lab. (a) and the relative map M_1 (b)

In this paper, we propose a system based on 2D LIDAR measurements capable of detecting changes in the environment over time and updating an existing map with a memory-limited algorithm taking into account localisation and measurement errors while neglecting highly dynamic obstacles as humans. Our system is fully developed through the Robot Operating System [6], using a 2D occupancy grid map representation due to their widespread use in the industrial field [7]. We are interested in taking as input a 2D occupancy grid map built at any moment with any available SLAM algorithm and producing a geometrically and temporally consistent representation of the environment suitable for any continued localisation algorithm based on occupancy grid maps.

Occupancy grid-based representations are vastly popular in robotics mapping thanks to their easy derivation from range sensor measurements [8]. Indeed, companies such

[1]Centro di Ricerca "E. Piaggio" e Dipartimento di Ingegneria dell'Informazione, Università di Pisa, Largo L. Lazzarino 1, Pisa, Italy.

[2]Soft Robotics for Human Cooperation and Rehabilitation, Fondazione Istituto Italiano di Tecnologia, via Morego, 30, Genova, Italy

[3]Proxima Robotics s.r.l., Via Olbia 20, Cascina, Pisa, Italy

This work was supported in part by the European Union's Horizon 2020 Research and Innovation Program under Grant Agreement Number 101017274 (DARKO), and in part by the Italian Ministry of Education and Research (MIUR) through the CrossLab Project (Departments of Excellence).

as MiR[1], BlueBotics[2], Gaussian Robotics[3], currently use 2D occupancy grid maps in their assets and this is an indicator of the industrial readiness level of these technology with respect to the others, and thus the need to make it even more robust in the long run. Since many industrial mobile robots are starting to use ROS and ROS2 as a developing system [9], we selected this framework to develop a new heuristic to allow such robots to autonomously navigate in a dynamic environment with a map that reflects the real-world configuration. Considering this framework, there are many open-source SLAM systems available to the robotics community to perform SLAM such as Gmapping [10], HectorSlam [11], Google Cartographer [**?**], and SLAM Toolbox [12]. However, they are not suitable for lifelong mapping and, therefore, they would be affected by aforementioned challenges of dynamic environments. Regarding the Slam Toolbox lifelong Mapping, since it is a graph-based method, it suffers from high computational costs and memory usage required for the graph node removal mechanism. In contrast, our approach exhibits reduced hardware consumption and memory usage.

II. RELATED WORKS

In occupancy grid approaches, space is divided into equally sized cells with associated binary random variable representing the probability that the cell is occupied by an obstacle (free, occupied, or unknown). Occupancy grid map algorithms compute approximate posterior estimates of the cell states and while all are derived from the Bayesian filters [13], they differ on the posterior calculation. While the problem of building a map of a static environment with occupancy grid approach has been already solved [8], their applicability in dynamic environment in long time operations should be further investigated. Several approaches have extended the occupancy grid map algorithms to deal with dynamic semi-static environments [14]–[17]. Such methods evaluate the occupancy of cells regardless of the kind of the object detected and hence avoid the multi-object detection and tracking issues related to data association [18]. However, they mainly focus on accelerating the speed of the mapping process but not on how to keep an initial occupancy grid map up to date for a long time. On the other hand, there exist grid based approaches oriented to lifelong mapping. A first class of such algorithms are dedicated to updating only local maps such as in [5], authors generate local maps containing only the persistent variations detected through a weighted recency averaging technique and perform local maps merging with Hough Transformation. However, they assume that dynamic objects are not allowed in the environment. In [19], the authors provide local maps built at different times and continuously update them online. However, this operation requires a large amount of memory to update the maps online. A solution to limit computational costs while keeping efficiency and consistency by pruning redundant local maps have been proposed in [20]. Unfortunately, those systems are not comparable with occupancy grid-based method like ours because they are based on different map representations. Other approaches are focused on life long mapping or life long SLAM but are not based on occupancy grid maps and hence not easily integrated and used in industrial systems. In [21], authors propose a new technique that employs a set of three maps to characterize the environment and implements a probabilistic feature persistence model to predict the state of obstacles and update the world model. Instead, in [22], they formalize each cell's occupancy as a failure analysis problem and contribute temporal persistence modeling (TPM), an algorithm for probabilistic prediction of the time that a cell in an observed location is expected to be "occupied" or "empty" given sparse prior observations from a task-specific mobile robot. However, even if the idea is promising, the approach is used to create a temporary map for motion planning and does not provide an updated map of the whole environment. On the other hand, many life-long SLAM systems in the literature provide the entire system dealing with a dynamic environment in long-term operations. Most of them rely on an internal structure such as a graph due to their incremental approach [23], [24]. However, even if they can provide an occupancy grid map for general navigation use, they take as input an existing graph structure and not a prior occupancy grid map.

III. NOTATIONS AND BASIC CONCEPTS

In this paper, we assume that we already have a map of the environment and only need to update it with changes detected by lidar measurements. The occupancy grid map divides the environment into individual cells $c_j(q)$ that contain information about the areas located at their associated positions $q \in \mathbb{R}^2$ in the space. Every cell c_j is described by a state that corresponds to the cell probability to be free (probability equal to 0), occupied (probability equal to 1), or unknown (otherwise). In our setup, the occupancy probability in the occupancy grid map is updated from laser point clouds. A laser range measurement $Z(k) = \begin{bmatrix} z_1(k) & \dots & z_n(k) \end{bmatrix}^T$ with n laser beams at time k, uniquely identifies a point cloud $P(k) = \begin{bmatrix} p_1(k) & \dots & p_n(k) \end{bmatrix}^T$, where $p_i(k) \in \mathbb{R}^2$ is the i-th hit point at time step k, i.e. the point in which the i-th beam hit an object. Given a laser scan measurement and the estimated robot's pose, the point cloud $P(k)$ (in fixed frame) can be easily computed through[4]:

$$p_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix} = \tilde{p} + z_i \begin{pmatrix} \cos(\theta_0 + i\,\Delta\theta + \tilde{\theta}) \\ \sin(\theta_0 + i\,\Delta\theta + \tilde{\theta}) \end{pmatrix}, \qquad (1)$$

where x_i, $y_i \in \mathbb{R}$ are the coordinates of the i-th measured hit point p_i, $\tilde{p} \in \mathbb{R}^2$ and $\tilde{\theta} \in \mathbb{R}$ are the robot estimated position and orientation, $\theta_0 \in \mathbb{R}$ is the angular offset of the first beam with respect to the orientation of the laser scanner and $\Delta\theta \in \mathbb{R}$ is the angular distance between two adjacent beams. Given a measured hit point p_i, we define the set of cells passed through by the i-th laser range measurement z_i corresponding to p_i as $C_{p_i} = \begin{Bmatrix} c_1, & \dots, & c_n \end{Bmatrix}$ where $c_n = c(p_i)$ is the cell associated to p_i.

[1]MiR,https://www.mobile-industrial-robots.com/about-mir/

[2]BlueBotics, https://bluebotics.com/

[3]Gaussian-Robotics, https://www.gaussianrobotics.com/

[4]to ease the notation, the time dependence k is omitted

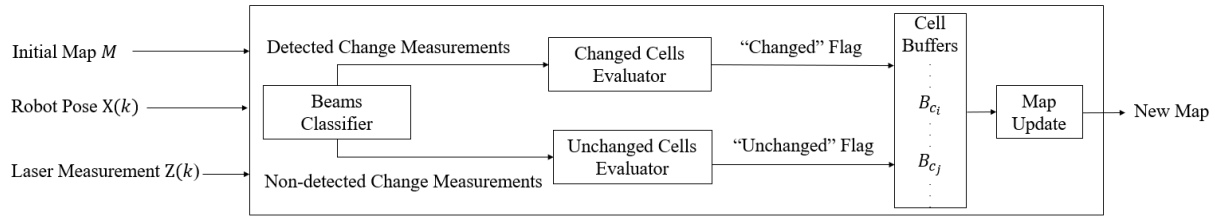
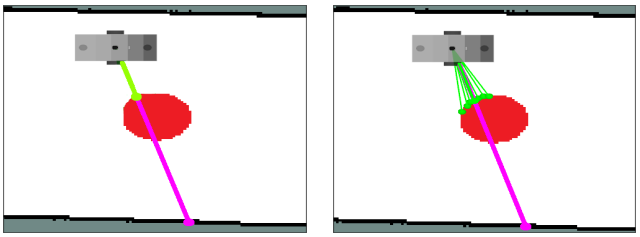


Fig. 2: System Overview: Measurements are classified as "detected change" or "non-detected change" with respect to the initial map by the *Beam Classifier*. Based on this classification, a rolling buffer B_{c_j} of each cell c_j is filled by the *Changed Cells Evaluator* and the *Unchanged Cells Evaluator* respectively through "changed" and "unchanged" flags. Finally, the state of the cells is updated if the number of "changed" flags in the buffer is higher than a given threshold.

IV. MAP UPDATE METHOD

In this section, the map updating developed method is described. Referring to the system overview in Fig.2, given an initial occupancy grid map M and the robot pose $X(k)$, the basic idea is to update the state of the cells according to the relevant changes in the environment detected by the laser measurements $Z(k)$. The system is built to detect both the removal or addition of static objects while neglecting the presence of dynamic obstacles, such as people, vehicles, or other robots, that can be sources of disturbances. Measurements are processed in two steps. First, the *Beams Classifier* analyses the sensor readings $z_i(k)$ and classifies them as "detected change measurement" or "non-detected change measurement" according to their discrepancy with respect to the initial map M as described in IV-A. Then the *Changed Cells Evaluator* and the *Unchanged Cells Evaluator* evaluate cells associated to measurements $z_i(k)$ with respect to those in the initial map M to confirm the type of detection. Indeed, confirmation procedures are required to deal with possible localisation errors and noise that can affect the measured information. Only for the purpose of the measurements process, we consider all the unknown cells in M as occupied to make it easier to detect the change. Indeed, by reducing the number of states from three (free, occupied, and unknown) to two (free and occupied), if a cell changes its state the new one is uniquely identified. To avoid false changes or undetected ones, we don't change the state of the cells based only on one measurement. Indeed, for each cell, $c_j \in C_{p_i}$, a rolling buffer, B_{c_j}, of a fixed dimension, N_b, is created and filled with the outcomes of the evaluator blocks at different time instant. The *Changed Cells Evaluator* takes as input only measurements $z_i(k)$ for which the *Beams Classifier* has "detected" a change and fills the buffer B_{c_j} for each associated cell, c_j, with a "changed" flag only if the change is confirmed as described in IV-B. Instead, the *Unchanged Cells Evaluator* analyses only measurements $z_i(k)$ for which the *Beams Classifier* provides a "non-detected" outcome and fills the buffer B_{c_j} for each associated cell, c_j, with an "unchanged" flag only if the evaluation is confirmed as described in IV-C. Finally, the state of evaluated cells c_j is changed from free to occupied (or vice versa) only if a sufficiently high number of "changed" flags can be found in the associated buffer, B_{c_j}, as described in IV-D.



(a) (b)

Fig. 3: Example of a measured hit point p_i (pink dot), the associated estimated hit point p_{exp_i} (case a)) and the associated set of expected measurements P_{exp_i} with $n_\text{exp} = 3$ (case b)) in green. In these maps, the red obstacle has been removed.

A. Beams Classifier

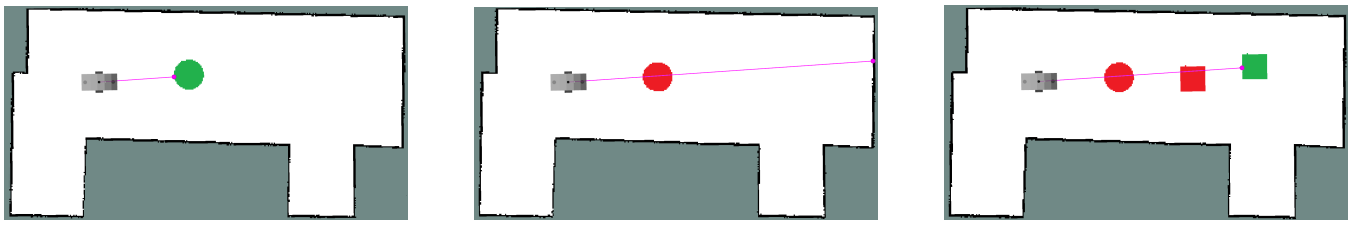
Given the measurement $z_i(k)$ and the current robot pose $X(k)$, the measured hit point $p_i(k)$ of the corresponding beam is computed as in (1). Such point must be compared with the expected measurement that the robot, in the same pose, would obtain if the environment fully correspond to the initial map M. We hence define the expected measurement $z_{\text{exp}_i}(k) \in \mathbb{R}^+$ as the expected value for the i-th range measurement $z_i(k)$ in $Z(k)$ computed from the initial occupancy grid map M. The corresponding expected hit point $p_{\text{exp}_i}(k) \in \mathbb{R}^2$ is computed as follows[1] (based on a ray casting approach [25]):

$$p_{\text{exp}_i} = \begin{pmatrix} x_{\text{exp}_i} \\ y_{\text{exp}_i} \end{pmatrix} = \tilde{p} + z_{\text{exp}_i} \begin{pmatrix} \cos(\theta_0 + i\,\Delta\theta + \tilde{\theta}) \\ \sin(\theta_0 + i\,\Delta\theta + \tilde{\theta}) \end{pmatrix}, \quad (2)$$

To detect a change, the euclidean distance between a measured hit point p_i and the expected one p_{exp_i} can be considered. Indeed, as Fig.3a[5] shows, a change leads to a discrepancy between these two points. Actually, to improve robustness, we consider a one-to-N comparison, i.e., each measured point p_i is not directly compared with p_{exp_i} but with a set P_{exp_i}, of $2n_\text{exp}$ points in map M built from p_{exp_i} itself. The expected hit points set P_{exp_i} is computed by adding a perturbation l to the angular components in the p_{exp_i} computation (2):

$$P_{\text{exp}_i} = \left\{ p_{\text{exp}_i}^{(-n_\text{exp})}, \quad p_{\text{exp}_i}^{(-n_\text{exp}+1)}, \quad \ldots, \quad p_{\text{exp}_i}^{(0)}, \quad \ldots, \quad p_{\text{exp}_i}^{(n_\text{exp})} \right\},$$

[5]In all the figures, the obstacles removed in the environment with respect to the initial map M are in red, while those added are green.



(a) A new obstacle has been added in the environment

(b) An obstacle has been removed from the environment

(c) Two obstacles have been removed, and a new obstacle has been added.

Fig. 4: Examples of "detected change" measurements caused by a change in the environment corresponding to added and/or removed obstacles. Laser beams are reported as pink lines, while the measured hit point is represented as a pink dot.

$$p_{\text{exp}_i}^{(l)} = \tilde{p} + z_{exp_i}^{(l)} \begin{pmatrix} \cos(\theta_0 + i\,\Delta\theta + \tilde{\theta} + l\,\Delta\theta_{\text{exp}}) \\ \sin(\theta_0 + i\,\Delta\theta + \tilde{\theta} + l\,\Delta\theta_{\text{exp}}) \end{pmatrix},$$

where $z_{\text{exp}_i}^{(l)} \in \mathbb{R}^+$ is defined in an analogous way as z_{exp_i}, $\Delta\theta_{\text{exp}}$ is the angular distance between adjacent expected beams, and $n_{\text{exp}} \in \mathbb{N}_0$ is a design parameter ($n_{\text{exp}} = 3$ in Fig.3b). Given this set, the i-th measurement detects a change if the minimum distance between the measured point p_i and expected points $p_{\text{exp}_i}^{(l)}$ belonging to P_{exp_i} is greater than a given threshold D_{th}. More formally, given a point cloud P associated to a laser scan measurement Z, the change for a measurement z_i is detected if the following holds:

$$\min_{p_{\text{exp}_i} \in P_{\text{exp}_i}} \|p_i - p_{\text{exp}_i}\|_2 > D_{\text{th}}. \tag{3}$$

It is worth nothing that the threshold can be chosen as a linear function of the distance z_i to take into account errors due to localisation and measurement noise.

B. Changed Cells Evaluator

The *Changed cells evaluator* module confirms if a change in a measurement z_i by the *Beam classifier*, corresponds to a change in the cells $c_j \in C_{p_i}$ associated to the measurement itself. The module stores a "changed" flag in the buffers of each cell with a confirmed change detection. In Fig.4 the two possible events that lead to a change in the environment are reported: the presence of a new obstacle (Fig.4a), and the obstacle removal (Fig.4b). More complex cases are a combination of these two (Fig.4c). In the next paragraphs, we will analyse in detail how our method distinguishes the two cases.

1) New obstacles detection: Let state(c_j) be a variable representing the occupancy state in the initial map M of the cell c_j. If a new obstacle is added, the state$(c(p_{\text{exp}_i}))$ of the cell associated to the expected hit point is "free" in the initial map M, while the measured hit point p_i associated with the "detected change" measurement z_i identifies an occupied cell. To confirm a new obstacle detection, we don't compare only the state of the cell concerning both the expected p_{exp_i} and the measured hit point p_i due to the localisation and noise errors, but we also analyse the state of the cells close to the examined one. Thus, given a measured hit point p_i, a new "changed" flag is added to the buffer of the $c(p_i)$ if the state of each cell in the initial map M, identified by a point belonging to a neighbourhood of p_i, is free, i.e. if:

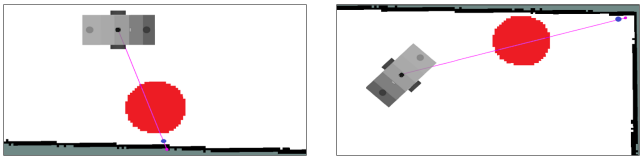
$$\text{state}(c_j(q)) = \text{free}, \quad \forall q \in \mathbb{R}^2 \mid \|q - p_i\|_2 < tol(z_i),$$

where $q \in \mathbb{R}^2$ is a point in the space and $tol \in \mathbb{R}^+$ is a function of the measured range as D_{th} in (3).

2) Detection of obstacles removal: In this case, considering the simplest case of Fig.4b, if a "detected change" measurement is not coherent with the map due to an obstacle disappearance, the laser beam passes through a certain number of occupied cells until it reaches an occupied cell in the initial map M. However, the measured hit point p_i is subject to localisation and noise errors leading to a comparison with a cell in the initial map that may be different from $c(p_i)$. Hence, to increase the algorithm robustness, we don't consider the final chunk of the laser beam by investigating only the cells between the robot's estimated pose and a point p_{th}, that lies on the laser beam (i.e. on the segment with extremities $\tilde{p}$ and p_i) and is at a fixed distance from p_i, as shown in Fig.5. We analyse only the cells $c_j \in C_{p_i}$ that satisfies:

$$\begin{aligned} &\|p_{\text{th}} - p_i\|_2 < \|p_{c_j} - p_i\|_2, \\ &p_{\text{th}} = m\tilde{p} + (1-m)p_i, \end{aligned} \tag{4}$$

where $m \in [0, 1]$ and $p_{c_j} \in \mathbb{R}^2$ is the point in the center of the cell c_j. If a cell c_j satisfies (4) and its state is occupied, a new "changed" flag is added to its buffer.



(a) (b)

Fig. 5: Example of change detection affected by localisation and noise errors. The hit point p_i (pink) can ends beyond (case a)) or before (case b)) the real obstacle, hence only cells up to p_{th} (blue) are evaluated for a change detection.

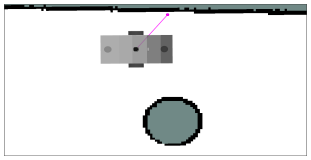
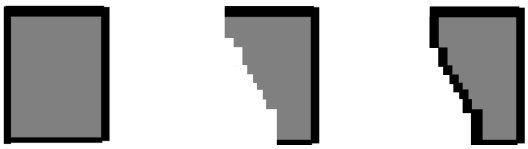


Fig. 6: Example of a "non-detected change" measurement affected by localisation and noise errors. The hit point (pink dot) cell does not correspond to an occupied cell in M.



(a) (b) (c)

Fig. 7: Last check of "free" cells: (a) A rectangular obstacle no longer in the real environment, (b) A part of the obstacle is removed obtaining new "free" cells, (c) The unknown frontier cells are marked as "occupied" during the last check to reconstruct the edge.

C. Unchanged Cells Evaluator

The *Unchanged Cells Evaluator* examines the "non-detected change" measurements provided by the *Beam classifier*. If the "non-detected change" measurement doesn't identify an environmental change, then all the map cells $c_j \in C_{p_i}$ associated to z_i didn't change their state with respect to the initial map M. If this happens, an "unchanged" flag is added to the buffer of those cells as follows:

1) Occupied cell: Since a change has not been detected by the *Beams Classifier*, the cell $c(p_i)$ associated to the hit point p_i should be occupied in the map M. However, due to localisation and noise errors, the measured hit point p_i could actually end up in a free cell in M (Fig.6) that should be close to occupied ones. Based on such observation, we may assume that given a "non-detected change" measurement, there will always be an occupied cell near the one associated to the hit point. Thus: if state$\big(c(p_i)\big) = free$ an "unchanged" flag is added to the buffers of occupied cells adjacent to $c(p_i)$ (if any). Otherwise, an unchanged flag is added to the buffer of the cell $c(p_i)$.

2) Free cells: Each cell $c_j \in C_{p_i} \setminus \{c(p_i)\}$ should be free in map M. However, due to localisation and noise errors the hit point p_i can end up beyond the obstacle and the procedure to recognize the "unchanged" cells is similar to that one described in IV-B.2:

1) Use ray casting to compute the cells that are passed through by the laser beam;
2) Discard the cells close to the measured hit point.
3) Mark as unchanged the remaining free cells and ignore the occupied ones.

D. Map Updating

The task of this module is to update the state of each cell $c_j \in C_{p_i}$ for all measurements z_i, analysing the number of "changed" flags in each cell buffer B_{c_j}. Let N_b be the size of the rolling buffers, let $n_{c_j} \in \big\{0, \quad \ldots, \quad N_\text{b}\big\}$ be a variable that counts the occurrences of the flag "changed" in B_{c_j}. The occupancy state of a map cell c_j is switched from free to occupied, or vice versa, only if n_{c_j} is larger than a given threshold. The threshold is a trade-off between the likelihood of mistakenly changing cells due to dynamic objects and the speed of the map update resulting in the speed of change detection. This threshold is a critical parameter to be set, with also N_b representing measurement memory. The bigger it is, the higher the number of flags it can memorise by remembering old measurements. Thanks to this approach, highly dynamic objects are first detected as changes in the measurements but then discarded in the map update since there are no sufficient "changed" flags in the buffer of cells interested by the moving object. During this phase, the "unknown" state for the cells of the initial map M is taken into account, and the last check on the updated cells is performed. This analysis is carried out to avoid situations in which an obstacle is not completely removed from the map, and therefore the localisation can be affected by the lack of an edge. These cases can occur in complex environments with objects very close to each other. For this purpose, for each new free cell, its adjacent cells are further investigated in map M, and if they are marked as "unknown", their state is changed to "occupied". A meaningful example of how this works is represented in Fig.7 where an obstacle is no longer in the environment, Fig.7a, but only part of the obstacle has been removed by updating the state of some of the occupied cells to "free", Fig.7b. During the further investigation, the state of the unknown frontier cells are finally updated to "occupied" and the edge is reconstructed, Fig.7c.

V. EXPERIMENTS

In this section, we present the results of our approach both in simulation and on real-world data. To provide a quantitative performance evaluation of the system, we compared our updated maps with ground-truth ones using several quantitative metrics. Moreover, we analysed the localisation errors with and without our updated maps using the Evo Python Package [26] in the simulation experiments. The code use to realize the reported experiments can be found at https://github.com/CentroEPiaggio/lidar_based_map_updating.

A. Map benchmarking metrics

In this work, we adopted the three following different metrics to evaluate the quality of our updated map with respect to the ground truth ones:

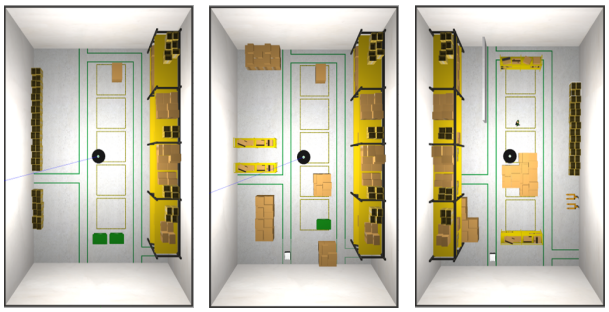
1) Cross-correlation (CC) [27]. The Cross-correlation metric measures the similarity between two maps based on means of occupancy probabilities of the cells. Let $m(q)$ be the occupancy probability of the cell that contains the point q in the map M. The cross-correlation coefficient between maps M and N, with n cells, is given by:

$$CC(M,N) = 100\left(\frac{\langle MN\rangle - \langle M\rangle \cdot \langle N\rangle}{\sigma(M)\cdot\sigma(N)}\right),$$

where

$$\langle M\rangle = \frac{1}{n}\sum_{m(q)\in M} m(q), \langle MN\rangle = \frac{1}{n}\sum_{\substack{m(q)\in M\\ n(q)\in N}} (m(q)\cdot n(q))$$

$$\sigma(M) = \sqrt{\frac{1}{n}\sum_{m(q)\in M}(m(q)-\langle M\rangle)^2}$$



(a) W_1 (b) W_2 (c) W_3

Fig. 8: Warehouse Gazebo environments.

2) Map score (MS) [27], [28]. The Map score metric compares two maps on a cell-by-cell basis:

$$MS(M,N) = 100\left(1 - \frac{1}{n}\sum_{\substack{m(q)\in M \\ n(q)\in N}} (m(q)-n(q))^2\right), \quad (5)$$

taking into account only cells that are occupied in at least one map to avoid favoring the map with large free space.

3) Occupied Picture-Distance-Function (OPDF) [29]. The Occupied Picture-Distance-Function metric compares the cells of a map M to the neighbourhood of the corresponding cells in the other map N and vice versa. The Occupied Picture-Distance-Function can be computed as:

$$OPDF_{as}(M,N) = 100\left(1 - \frac{1}{no \cdot r}\sum_{i=1\ldots no}^{no} d_i\right), \quad (6)$$

where no is the number of occupied map cells in M, d_i is the minimum between the search space amplitude r (e.g., a rectangle of width *wscpace* and height *hsspace*, $r = \sqrt{wsspace^2 + hsspace^2}$) and the Manhattan-distance of each occupied cell of the first map M to the closest occupied cell on the second map N. Since the distance function in (6) is not symmetric, we consider the average of distances from M to N and from N to M:

$$OPDF(M,N) = \frac{OPDF_{as}(M,N) + OPDF_{as}(N,M)}{2}.$$

B. Simulation experiments

The simulation experiments were performed on a laptop with an Intel Core i7-10750HCPU, 16 GB of RAM, and Ubuntu 18.04. We simulated an industrial warehouse of 290 m^2 using the models provided by Amazon Web Services Robotics [30] to build a world on the Gazebo simulator [31]. The robot used is the Robotnik XL-Steel platform equipped with two SICK s300 lidars [32]. We built four different versions of the same environment by increasing the changes to simulate how the placement of goods within a warehouse can change over time. Due to space constraints, we have omitted the last world simulation since we obtained similar results and would therefore be redundant. In the first environment W_1, Fig. 8a, the robot was tele-operated to build an adequate initial map M_1. In the other environments, reported in Fig.s 8b, 8c, with environments $W_i, i \in \{2,3\}$ respectively, the robot autonomously performed the same predefined trajectory simulating a material transport in the warehouse. In each scenario W_i, with $i \in \{2,3\}$, the robot used the map M_{i-1} as the initial map to localise itself, and generated the updated map M_i with the proposed updating method. To evaluate the quality of the method a ground-truth maps G_i for each run$_i$ with $i \in \{1,2,3\}$ has been obtained with Slam Toolbox. Adaptive Monte Carlo Localisation (AMCL) [33] from ROS have been used for robot localisation. It is worth noting that, the same map use-and-update approach and the evaluation metrics have been used for a car parking scenario with humans; outcomes are visible in the multimedia material and not reported here for space limitations. Finally, an experiment on a real robot have been performed to test the updating map method during navigation in a partially-mapped place.

1) Updating Performance: The first map M_1 related to W_1 and the planned trajectory are reported in Fig. 9[6]

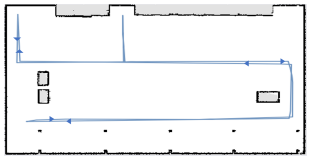
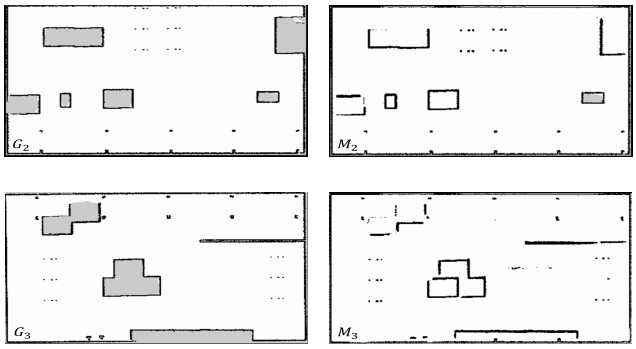


Fig. 9: Autonomous trajectory path (blue) in first map M_1



(a) Ground truth. (b) Proposed Approach.

Fig. 10: Map comparisons

	W_2		W_3	
	M_1/G_2	M_2/G_2	M_1/G_3	M_3/G_3
CC (%)	45.05	69.26	31.78	60.61
MS (%)	48.35	70.66	36.66	61.70
OPDF (%)	66.08	95.64	53.25	91.32

TABLE I: Quantitative maps evaluation.

A qualitative result of our updating method is depicted in Fig.10 comparing ground-truth maps $G_i, i \in \{2,3\}$ and corresponding updated maps $M_i, i \in \{2,3\}$. Objects detected in the first world and still present in the following ones contain "unknown" states cells (in grey), while obstacles added in subsequent worlds and not present in previous ones contain free cells (in white). Indeed, since they are related to objects' internal parts, they are physically undetectable by the robot's lidar. Hence, we didn't consider updating those cells relevant for autonomous navigation purposes. It is worth noting how obstacles no more present in the environment have been entirely removed in the updated

[6]Maps have all dimension 13,95mx20,9m with 5 cm of resolution.

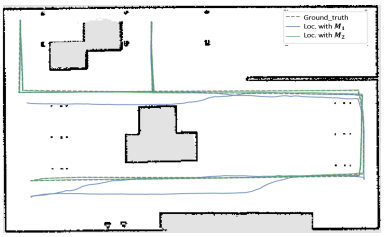


Fig. 11: Estimated AMCL pose comparison in W_3. Blue localisation with M_1, green with last updated map, dashed ground truth

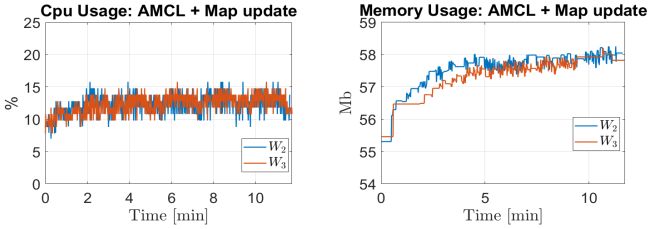
maps. This qualitative comparison shows that our method detects environmental changes, and each provided map reflects the configuration during the simulation. Table I shows the quantitative results obtained by comparing both the initial map M_1 and our updated maps $M_i, i \in \{2,3\}$ with respect to the ground-truth maps $G_i, i \in \{2,3\}$ using the metrics described in V-A where a 100% score is a full correspondence of the two maps. A map comparison between M_{i-1} and G_i is performed to quantify differences between current and previous environment (first column for each W_i). According to each metric, the updated map M_i is always better than the initial map M_{i-1} of the run when compared with the ground-truth. However, as shown in the second column of each W_i, the first two metrics are affected by not having updated the obstacles internal cells, while this does not affect the third metric that reports more truthful results. It is important to underline that the robot predefined trajectory is not intended to explore the warehouse area but only to transport material in it and hence measurements can miss changes in the environment that are not visible along the robot displacement.

2) Localisation Performance: To evaluate improvements in localisation performance thanks to the use of updated maps, we compared the AMCL pose estimate based on both the initial map M_1 and the last available updated map with the reference ground truth obtained from the simulator. Fig. 11 shows the comparison between the three trajectories in W_3: the trajectory performed based on localisation with map M_1 is in blue, the one obtained with map M_2 is in green while the ground truth is the dashed line. As expected, using an updated map, the localisation error is greatly reduced. On the other hand, employing an outdated map, the robot has bumped into some objects. Quantitative results of the localisation performance are reported in Tab. II where as shown in the second line the localisation is drastically improved thanks to the use of the updated map M_2.

		Max	Mean	Median	Min	RMSE	SSS	Std
W_3	M_1	1.05	0.31	0.08	0.03	0.50	113.90	0.35
	M_2	0.12	0.05	0.05	0.03	0.05	1.57	0.01

TABLE II: Localisation Performance comparison in W_3 using both the initial map M_1 and the last updated map M_2. RMSE = Root Mean Square Error, SSS = Sum Squared Error

3) Hardware Resource Consumption: In this paragraph, we report the hardware resources in terms of CPU percentage and memory MB usage (computed through the ROS



(a) CPU Usage (b) Memory Usage

Fig. 12: Resource Consumption

package "cpu_monitor"[7] in the map updating and localisation phases. Fig. 12a shows the percentage of CPU used while Fig. 12b the one of memory usage for the map update. The computational load is mainly due to the laser scan measurements processing and it can be reduced by discarding some range measurements from the processed laser scan. The memory usage depends on the number of changed cells and the size N_b of the buffers. Thus, the size of the environment determines an upper bound on the required memory. In the considered scenario and with a buffer size of 10, if all the cells changed their state the memory usage would be at most 150 MB. Thus, the proposed memory-limited solution is suitable for life-long operation scenarios.

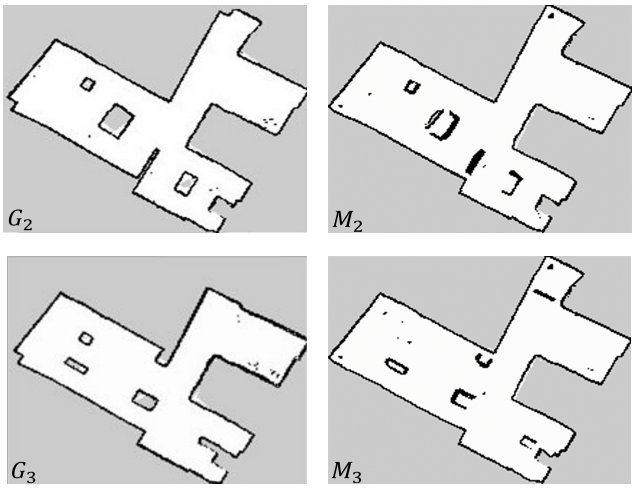
C. Real Experiments

To demonstrate the real-world applicability of the proposed method, we conducted a set of experiments in the lab, Fig. 1a, with a Summit-XL-Steel mobile platform with two 2D-LIDAR Hokuyo-UST-20LX. We reproduced four different environments with changes in obstacles position. Due to space contraints, we have omitted the last scenario since we obtained similar results and would therefore be redundant. The testing environment is approximately 80 m^2. The robot built the initial map M_1 (Fig. 1b) and the ground-truth ones $G_i, i \in \{2,3\}$ (Fig. 13a) where obtained through Slam Toolbox during a tele-operation navigation at $0,15m/s$. To test our algorithm and equally compare the data in the different worlds, bag files with registered sensor measurements and odometry topics during the robot navigation have been used. Performance on map update and hardware resource consumption have been quantified as in the simulations. For the localisation performance, since no ground-truth external tracking system was available, it was not possible to provide valid and scientifically acceptable localisation performance results.

1) Updating Performance: The qualitative and quantitative results are shown in Figure 13[8] and Table III. Considerations in Sub-section V-B.1 regarding the metrics results are still valid here. However, even in the case of noisy real data, with greater localisation errors, the proposed method always produces updated maps that are better than the initial one by comparing them with ground truths and showing to be able to remove and add obstacles in real time without shifting or removing the walls.

[7]cpu_monitor, https://github.com/alspitz/cpu_monitor

[8]Maps have all dimension 10,5mx8,55m with 5 cm of resolution.



(a) Ground truth. (b) Proposed Approach.

Fig. 13: Map comparisons

	W_2		W_3	
	M_1/G_2	M_2/G_2	M_1/G_3	M_3/G_3
CC (%)	69.69	75.98	63.77	68.80
MS (%)	54.63	64.53	51.44	57.36
OPDF (%)	84.61	95.05	78.92	90.77

TABLE III: Quantitative maps evaluation.

2) Hardware Resource Consumption: The CPU and memory usage are lower respectively from 15% to 5% and from 55-58Mb to 51-54Mb with respect to the simulated environments because of the decrease in no. of updated cells since the real environment is smaller than the simulated one.

VI. CONCLUSIONS

In this paper, we proposed a method based on occupancy grid maps to deal with dynamic environments for long term operations. The goal was to update the map robustly with respect to localisation and measurement errors, neglecting humans, and limiting memory storage. The approach has been tested in simulation and with real world experiments. The updated maps do not show signs of drift or inconsistency even when the localisation error is relatively large, moreover they reflect the environment configuration and increase the AMCL localisation performance in simulations. In both cases memory storage has been shown to be limited. As future work, we plan to validate the localization performance in a real environment with an external tracking system.

REFERENCES

[1] T. A. Styleintelligence, "Market report: Agv & amr robotics 2021," 2021, november 2021. [Online]. Available: https://www.styleintelligence.com/collections/the-reports/products/agv-amr-robotics-2021

[2] T. Chong, *et al.*, "Sensor technologies and simultaneous localization and mapping (slam)," *Procedia Computer Science*, vol. 76, pp. 174–179, 2015.

[3] M. Dymczyk, *et al.*, "Map summarization for tractable lifelong mapping," in *RSS Workshop*, 2016.

[4] D. Meyer-Delius, *et al.*, "Temporary maps for robust localization in semi-static environments," in *2010 IEEE/RSJ Int. Conf. Intell. Robots Syst.* IEEE, 2010, pp. 5750–5755.

[5] F. Abrate, *et al.*, "Map updating in dynamic environments," in *ISR 2010 (41st International Symposium on Robotics) and ROBOTIK 2010 (6th German Conference on Robotics)*. VDE, 2010, pp. 1–8.

[6] M. Quigley, *et al.*, "Ros: an open-source robot operating system," vol. 3, 01 2009.

[7] F. Amigoni, *et al.*, "A standard for map data representation: Ieee 1873-2015 facilitates interoperability between robots," *IEEE Robotics Automation Magazine*, vol. 25, no. 1, pp. 65–76, 2018.

[8] S. Thrun, *et al.*, *Probabilistic robotics*. Cambridge, Mass.: MIT Press, 2005.

[9] R. Tellez, "Top 10 ros-based robotics companies in 2019," 2019, july 22, 2019. [Online]. Available: https://www.therobotreport.com/top-10-ros-based-robotics-companies-2019/

[10] G. Grisetti, *et al.*, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE transactions on Robotics*, vol. 23, no. 1, pp. 34–46, 2007.

[11] S. Kohlbrecher, *et al.*, "A flexible and scalable slam system with full 3d motion estimation," in *Proc. IEEE Int. Symp. Saf. Secur. Rescue Robot.(SSRR)*. IEEE, November 2011.

[12] S. Macenski and I. Jambrecic, "Slam toolbox: Slam for the dynamic world," *J. Open Source Softw.*, vol. 6, no. 61, p. 2783, 2021. [Online]. Available: https://doi.org/10.21105/joss.02783

[13] S. Thrun, "Robotic mapping: a survey," 2003.

[14] D. Meyer-Delius, *et al.*, "Occupancy grid models for robot mapping in changing environments," in *AAAI*, 2012.

[15] Q. Baig, *et al.*, "A robust motion detection technique for dynamic environment monitoring: A framework for grid-based monitoring of the dynamic environment," *IEEE Robot. Autom. Mag.*, vol. 21, no. 1, pp. 40–48, 2014.

[16] D. Nuss, *et al.*, "A random finite set approach for dynamic occupancy grid maps with real-time application," *The Int. J. Rob. Res.*, vol. 37, no. 8, pp. 841–866, 2018.

[17] J. Huang, *et al.*, "An online multi-lidar dynamic occupancy mapping method," in *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2019, pp. 517–522.

[18] A. Llamazares, *et al.*, "Detection and tracking of moving obstacles (datmo): A review," *Robotica*, vol. 38, no. 5, p. 761–774, 2020.

[19] P. Biber and T. Duckett, "Dynamic maps for long-term operation of mobile service robots," in *Robotics: Science and Systems*, 2005.

[20] N. Banerjee, *et al.*, "Lifelong mapping using adaptive local maps," in *2019 European Conference on Mobile Robots (ECMR)*. IEEE, 2019, pp. 1–8.

[21] M. L. Pitschl and M. W. Pryor, "Obstacle persistent adaptive map maintenance for autonomous mobile robots using spatio-temporal reasoning*," in *2019 IEEE 15th Int. Conf. Autom. Sci. Eng.(CASE)*, 2019, pp. 1023–1028.

[22] G. Tsamis, *et al.*, "Towards life-long mapping of dynamic environments using temporal persistence modeling," in *2020 25th International Conference on Pattern Recognition (ICPR)*, 2021, pp. 10 480–10 485.

[23] M. T. Lázaro, *et al.*, "Efficient long-term mapping in dynamic environments," in *2018 IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*. IEEE, 2018, pp. 153–160.

[24] M. Zhao, *et al.*, "A general framework for lifelong localization and mapping in changing environment," in *2021 IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*. IEEE, 2021, pp. 3305–3312.

[25] J. Amanatides and A. Woo, "A fast voxel traversal algorithm for ray tracing," *Proceedings of EuroGraphics*, vol. 87, 08 1987.

[26] M. Grupp, "evo: Python package for the evaluation of odometry and slam." https://github.com/MichaelGrupp/evo, 2017.

[27] O. Sullivan, "An empirical evaluation of map building methodologies in mobile robotics using the feature prediction sonar noise filter and metric grid map benchmarking suite," Master's thesis, University of Limerick, 2003.

[28] M. C. Martin and H. P. Moravec, "Robot evidence grids." Carnegie-Mellon Univ Pittsburgh Pa Robotics Inst, Tech. Rep., 1996.

[29] K. Baizid, *et al.*, "Vector maps: A lightweight and accurate map format for multi-robot systems," in *Int. Conf. Intell. Robots Syst.* Springer, 2016, pp. 418–429.

[30] A. W. S. Robotics, "aws-robomaker-small-house-world," https://github.com/aws-robotics/aws-robomaker-small-house-world.

[31] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ Int. Conf. Intell. Robots Syst. (IEEE Cat. No.04CH37566)*, vol. 3, 2004, pp. 2149–2154.

[32] RobotnikAutomation, "Robotnik xl-steel simulator," https://github.com/RobotnikAutomation/summit_xl_sim.

[33] D. Fox, *et al.*, "Monte carlo localization: Efficient position estimation for mobile robots," 01 1999, pp. 343–349.