Map updating in dynamic environments

Fabrizio Abrate, Basilio Bona, Marina Indri, Stefano Rosa and Federico Tibaldi
Politecnico di Torino, Italy

Abstract

While building maps when robot poses are known is a tractable problem requiring limited computational complexity, the simultaneous estimation of the trajectory and the map of the environment (known as SLAM) is much more complex and requires many computational resources. Moreover, SLAM is generally peformed in environments that do not vary over time (called *static* environments), whereas real applications commonly require navigation services in changing environments (called *dynamic* environments). Many real robotic applications require updated maps of the environment that vary over time, starting from a given known initial condition. In this context classical SLAM approaches are generally not directly applicable: such approaches only apply in static environments or in dynamic environments where it is possible to model the environment dynamics. We are interested here in long-term mapping operativity in presence of variations in the map, as in the case of robotic applications in logistic spaces, where rovers have to track the presence of goods in given areas. In this paper we propose a methodology that is able to detect variations in the environment, generate a local map containing only the persistent variations and finally merge the local map with the global one used for localization.

1 Introduction

Localization and map building are closely linked problems, and are often referred to as simultaneous localization and mapping (SLAM). While building maps when robot poses are known is a tractable problem with limited computational complexity, in the SLAM case the uncertainty in measures and in robot pose estimates makes the problem much more complex.
Among all the possible strategies, two main solutions for the SLAM problem exist.
The first one models the environment using features and manages the associated uncertainty by the Extended Kalman Filter (EKF). This approach is extremely compact and its computational cost has been considerably improved. For instance in [11] an EKF SLAM algorithm is described where the computational complexity per step is reduced from $O(n^2)$ to $O(n)$, and the total cost of SLAM is reduced from $O(n^3)$ to $O(n^2)$, being n the number of features.
The main drawback of this technique is its difficulty to model different types of environments, due to the limited set of feature models available.
The second solution [13] solves the SLAM problem using particle filters, which are sequential Monte Carlo methods to estimate posterior probability distribution functions. This approach is suitable to be used in unstructured environments, especially when combined with a grid representation of the considered environment, but it suffers from high computational cost due to the usually high number of particles required to approximate posteriors.
Some attempts have been made in order to reduce the computational cost of this technique. In [10] a new solution to the grid-based SLAM problem based on the stochastic search of the best pose estimate is proposed. This solution decreases the computational cost compared to other solutions. In [7] adaptive techniques that reduce the number of particles in Rao-Blackwellized particle filters are used.
Many real applications require updated maps of the environment that vary over time, starting from a given initial condition. This is for instance the case of robotic applications in logistic spaces, where robots have to track the presence of goods in a certain area. The goods are stored in appropriate places, but during the day they can be removed and substituted by other items many times.
This paper deals with the problem of keeping the map updated, in order to guarantee the robots localization; specific goods tracking procedure are not investigated.
The initial map can be a-priori known or built using a classical SLAM algorithm. In the latter case, the environment is supposed to be static during the initial SLAM process. Then the environment starts to change, and the map needs to be updated, as modifications are sensed by the robot.
During the map update process, there are mainly two issues to be considered:

1. a long-term operativity is required;
2. the algorithm performing the map update has to be computationally light and to use limited memory.

The first item is mandatory since in real applications map variations may occur continuously for a long period of time (even an entire day), and the map updating process should be carried out as long as possible without diverging.
The second item is very important when the map updating process has to be carried out in parallel with other tasks, (e.g., team coordination and planning, surveillance).
A combined algorithm for map update and robot localization is proposed in [14], while a method for updating the map dynamically during the process of localization is developed in [9]. This method seems very promising but its experimental validation is still only partial. The problem of grid mapping through belief propagation is investigated in [12], assuming that good estimates of the robot position are available. A spatial Markov random field model is applied to a minefield mapping. Two things are required: the list of the type of objects that populate the environment and the a-priori knowledge about where the objects are positioned. Finally [5] introduces a dynamic map representation for mobile robots that adapts continuously over time. It

solves the stability-plasticity dilemma (the trade-off between adaptation to new patterns and preservation of old patterns) by representing the environment over multiple timescales simultaneously.
In this paper we propose a methodology that is able to:

- detect variations in the environment;
- generate a local map containing only the persistent variations;
- merge the local map with the global one used for localization.

The variations of the environment are detected using a technique called *weighted recency averaging*, while the local maps are merged employing the Hough transform. Section 2 states the problem considered in this paper while Section 3 describes the approach designed to solve the problem. Sections 4 and 5 show the results obtained in simulation and experimental tests, while Section 6 draws some conclusions.

2 Problem Formulation

A mobile robot, endowed with a laser rangefinder, is supposed to be correctly localized with respect to the available environment map. Let its estimated pose at time t be denoted by

$$\hat{p}(t) = \{\hat{x}(t), \hat{y}(t), \hat{\theta}(t)\}. \tag{1}$$

By the expression *correctly localized robot* we indicate a robot that is in the *position tracking* state, as defined in [2] and [3].
An occupancy grid map of the environment (used in the localization algorithm to track the robot position over time) is available to the robot. Such a map could have been manually created or previously built by a SLAM algorithm.
At discrete instants k the environment changes, and consequently the robot has to modify its map, to take into account the variation. We call this phase a Δ-*mapping* step.
We define the set of new maps collected up to time k as

$$\mathcal{M}(K) = \{M_k\},\ k = 0, \dots, K.$$

M_0 is the initial map, obtained by the SLAM procedure. The goal of the algorithm developed in the next section is to provide an estimate $\hat{M}_k$ of the map at each time step k.

3 The Approach

The architecture of the approach can be represented by separate functional blocks (as shown in Figure 1).
The Δ-Awareness block implements the technique used to detect when a change has occurred in the map, and consequently communicates to the robot that it is time to enter the so called Δ-*mapping* phase.
The *Store-scan* block decides which scans acquired during the Δ-mapping phase are suitable to create a local map containing the variations.
The *Alignment* block registers in a consistent way the set of measurements frames (range scans) collected by the *Store-scan* block. The approach maintains all the local frames of data as well as the relative spatial relationships between local frames. The adopted approach is described in [8].
The *Map Merge* block merges the output of the *Alignment* block at time k with the map $\hat{M}_{k-1}$.
The most relevant blocks are detailed in the next subsections.

Figure 1: The architecture of the approach.

3.1 Δ-Awareness

The Δ-Awareness block detects persistent variations in the environment, using a technique called *weighted recency averaging*, which is normally applied in problems of tracking non-stationary processes.
An example of application of this technique can be found in recovering from localization failures [13], where it is used to calculate the empirical measurement likelihood and maintain short-term and long-term averages of this likelihood, deciding when to randomly inject particles to the filter.
In our setting, the weighted recency averaging is employed to recognize changes in the environment, under the hypothesis that the robot is correctly localized and never kidnapped.
In particular, we call $w_{avg}(t)$ the weight of the position hypothesis with the highest likelihood, and we calculate the short-term likelihood and the long-term likelihood as

$$\begin{array}{rcl} w_{slow}(t+1) & = & w_{slow}(t) + \alpha_{slow}(w_{avg}(t) - w_{slow}(t)) \\ w_{fast}(t+1) & = & w_{fast}(t) + \alpha_{fast}(w_{avg}(t) - w_{fast}(t)) \end{array} \tag{2}$$

where the parameters α_{slow} and α_{fast} are decay rates for the filters that estimate the long-term and short-term averages. As a rule of thumb, we should choose $\alpha_{slow} \ll$

α_{fast}, noticing that these parameters influence directly how quickly the Δ-Awareness block perceives a variation occurred in the environment.
The $w_{avg}(t)$ is in general a non-stationary process, since at least one of its parameters (e.g. the mean value) changes over time.
In Figure 2 it is shown a comparison between the trend of the $w_{avg}(t)$ over time when no modifications in the map occur (a), and when the robot passes near an area with a variation (b). The sudden changes of $w_{avg}(t)$ in the latter case allow to use such process to detect variations.

The divergence between the short-term and the long-term average of the measurement likelihood is considered in the computation of the following *divergence ratio* $r_d(t)$

$$r_d(t) = 1 - \frac{w_{fast}(t)}{w_{slow}(t)} \tag{3}$$

which is one of the indexes considered by the Store Scan block to decide if a laser scan should be discarded or not, as discussed in Subsection 3.2.

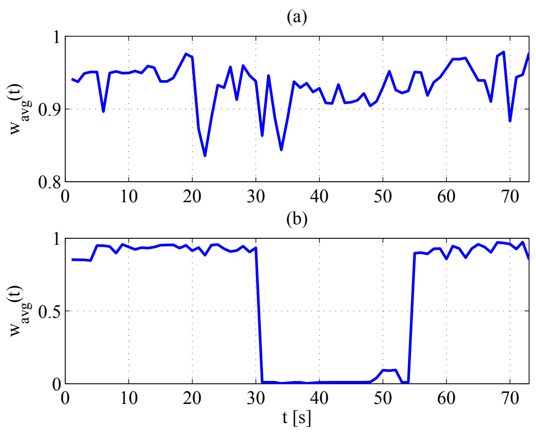


Figure 2: Comparison between the trend of the $w_{avg}(t)$ over time when no modifications in the map occur (a), and when (b) the robot passes near a variation.

3.2 Store Scan

The purpose of the Store Scan block is to select the laser scans suitable for building the local updated sub maps. These scans are stored in a set called $\mathcal{S}(k)$.
In order to maximize the probability of storing scans when modifications in the map have occurred, if $r_d(t) > 0$ a scan is selected with a probability given by $v < r_d(t)$, where v is a uniformly distributed random variable $U(0, 1)$.
Figures 3 and 4, respectively, clarify what happens when there are no modifications in the map and when a robot perceives a variation in the environment.

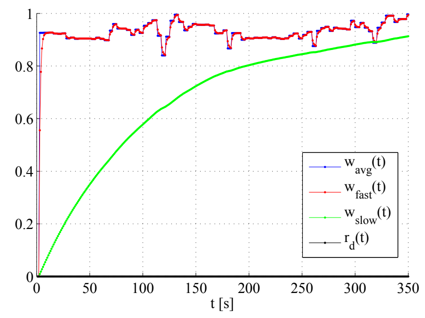


Figure 3: Trend of $w_{avg}(t)$, $w_{fast}(t)$, $w_{slow}(t)$, $r_d(t)$ in absence of modifications in the environment.

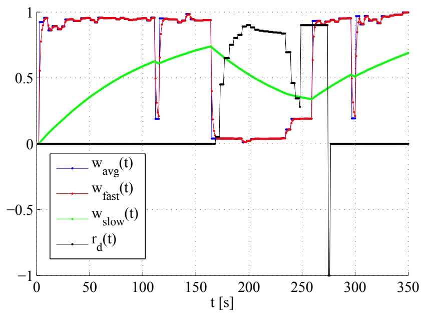


Figure 4: Trend of $w_{avg}(t)$, $w_{fast}(t)$, $w_{slow}(t)$, $r_d(t)$ in presence of modifications in the environment.

In Figure 3 the values taken by $w_{fast}(t)$ are higher than those taken by $w_{slow}(t)$; from (3) it follows that $r_d(t)$ is negative (in Figure 3 it has been set to zero) and the Δ-mapping process does not begin. In Figure 4 the values assumed by $w_{fast}(t)$ are lower than those assumed by $w_{slow}(t)$ and hence $r_d(t)$ is greater than zero.
The Δ-mapping process begins when $r_d(t)$ becomes positive for the first time, and ends when $r_d(t)$ falls again below zero.
Unfortunately we experienced that this method suffers from a couple of problems.
It can be observed that even if the environment does not change, the *divergence ratio* may increase in specific situations, such as when the rotational velocity of the robot is greater than zero, causing the beginning of the Δ-mapping process also when the environment has not changed.
The second problem is that even if the Δ-mapping process has correctly begun due to a variation in the environment, the quality of the scans acquired by the laser rangefinder could be insufficient.
The accuracy of the scans is related to the precision of the robot pose estimation, in particular to its heading estimation. An interesting example in the context of EKF-SLAM is presented in [4], where it is claimed that a significant source of inconsistency is to be found in heading variance which, if large, can cause divergence in just a few updates. In our case, scans acquired with a wrong heading may lead to bad-aligned sub-maps. Moreover, the problem of a bad

quality of the scans can be related to the first highlighted matter, i.e. when the robot is rotating.
In fact it frequently happens that the robot encounters map variations while turning, hence when its rotational velocity is greater than zero. In this situation, even if the map has changed, some acquired scans may have a wrong heading, and therefore they should be discarded.
The proposed solution for a proper scan discarding is discussed in the next subsection.

3.2.1 Scan discarding based on the evaluation of N_{eff}

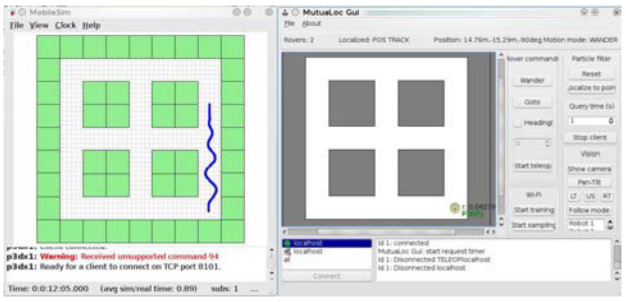
The local degeneracy of the particle filter algorithm is measured by the *effective sample size* N_{eff} of the particle filter defined as

$$N_{\text{eff}} = \frac{N_s}{1 + \text{Var}(w_t^{*i})} \tag{4}$$

where N_s is the number of particles and w_t^{*i} is the true weight. N_{eff} cannot be exactly calculated, but an estimate can be obtained by computing

$$\hat{N}_{\text{eff}} = \frac{1}{\sum_{i=1}^{N_s} (w_t^i)^2} \tag{5}$$

A relation has been empirically found between the local degeneracy of the particle filter and the moments in which the robot is turning. Three cases have been considered to investigate such a relation. In the first one we compute the values of N_{eff} while the robot is going straight and some environment changes occur. In the second one the robot turns with a constant rotational velocity (without any environmental modification), while in the third one it follows the blue (not straight) path shown in Figure 5.



p3dx1: Warning: Received unsupported command 94
p3dx1: Ready for a client to connect on TCP port 8101.
Time: 0:0:12:05.000 (avg sim/real time: 0.89) subs: 1
Id 1: connected
MutuaLoc Gui: start request timer
Id 1: Disconnected TELEOPlocalhost
Id 1: Disconnected localhost

Figure 5: Path followed by the simulated robot to evaluate N_{eff} in the third case. On the left there is the simulated robot and environment, on the right our graphical user interface.

The results of computing N_{eff} in the three examples are shown in Figure 6, in blue, red and black, respectively.
The red and black plots show that the robot turning decreases the value of N_{eff}, whereas the variation in the map while the robot is *not* turning does not influence its value. We can thus argue that the evaluation of N_{eff} allows us to reject those scans acquired while the robot is turning, which can be bad aligned with high probability.

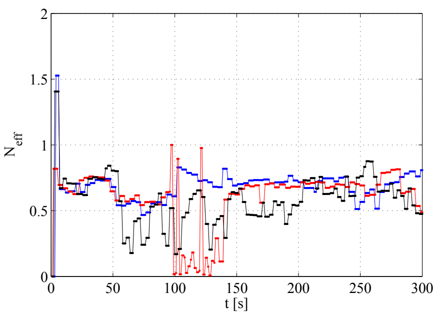


Figure 6: Comparison of the N_{eff} trend.

3.3 Scan Alignment

The Scan Alignment block produces a local map performing a consistent registration of the collection of scans contained in $\mathcal{S}(k)$. The approach maintains all the local frames of data as well as the relative spatial relationships between local frames, modeled as random variables and derived from matching pairwise scans or from odometry. Then all the spatial relations are combined using the $Rmap$ algorithm (see [1]).
The output of this block is a consistently aligned map S_k.

3.4 Map Merge

The Map Merge block receives the aligned map S_k provided by the *Scan Alignment* block and merges this map with $\hat{M}_{k-1}$, which is the map the robot is currently using for localization. The output of this merge process is $\hat{M}_k$.
We adopted the algorithm proposed in [6], whose theoretical foundations are briefly recalled in the next subsection.

3.4.1 Theoretical background about Map Merging

We assume that a grid map M is a matrix with r rows and c columns. Each cell $M(i, j)$ may contain three different values, indicating whether the cell is free, occupied, or if its status is unknown.
Given two maps $M1$ and $M2$, the goal of map merging is to find a rigid transformation T so that the two maps can be overlapped. The transformation $T = T(\Delta x, \Delta y, \phi)$ is the combination of a rotation ϕ, followed by a translation along the x and y axis of magnitude Δx and Δy, respectively.
The overall transformation T is computed in two separate steps. First the rotation ϕ is determined, and then the translations Δx and Δy are deduced. In order to do this, Hough transform is applied to detect lines expressed in polar coordinates (i.e. ρ and θ). Line detection is performed using the Discretized Hough transform (DHT), that discretizes the domain for ρ and θ, so that the DHT can therefore be represented by a matrix with ρ_S rows and θ_S columns. In addition it is also necessary to set a bound for ρ, while θ is naturally bounded to $[0, 2\pi)$. The DHT can be applied to detect lines in an occupancy grid map M, by converting it

into a binary image. The conversion can be performed setting all occupied cells to black, and all other cells to white, for example. If M is a grid map, we indicate its DHT with HT_M. Given HT_M we define its associated Hough Spectrum as the following signal:

$$HS_M(l) = \sum_{i=1}^{\rho_S} HT_M(i,l)^2,\ 1 \le l \le \theta_S$$

Translations of the Hough spectra correspond to rotations of the associated map. The computation of the circular cross correlation gives information about how the maps have to be rotated in order to be overlapped. Formally, if HS_{M1} and HS_{M2} are two Hough spectra with the same sampling period, their circular cross correlation CC_{M1M2} is a signal with the same sampling period defined as follows:

$$CC_{M1M2} = \sum_{i=1}^{\theta_S} HT_{M1}(i)HT_{M2}(i+l),\ 1 \le l \le \theta_S \quad (6)$$

Local maxima in the spectra cross correlation reveal how $M2$ should be rotated in order to align it with $M1$. The proposed algorithm therefore extracts a set of n local maxima (n being a specified parameter), and returns n transformations.
Given a candidate rotation ϕ_i , the corresponding translations Δx^i , Δy^i can be in principle easily determined. Let $M3$ be the map obtained rotating $M2$ of ϕ_i , i.e.

$$M3 = T(0,0,\phi_i)M2. \quad (7)$$

Translations needed to overlap $M3$ to $M1$ can be obtained computing the bidimensional correlation in the following way. First we compute the *X-spectrum* of a binary image M, having r rows and c columns, as follows:

$$SX_M(j) = \begin{cases} \sum_{i=1}^{r} M(i,j) & 1 \le j \le c \\ 0 & \text{otherwise} \end{cases} \quad (8)$$

Similarly, the *Y-spectrum* of image M is defined as:

$$SY_M(j) = \begin{cases} \sum_{i=1}^{c} M(i,j) & 1 \le j \le r \\ 0 & \text{otherwise} \end{cases} \quad (9)$$

Given SX_{M1} and SX_{M3}, Δx^i can be easily inferred by looking at the global maximum of the cross correlation between them, defined as

$$CCX_{M1M3}(\tau) = \sum_{l=-\infty}^{+\infty} SX_{M1}(l+\tau)SX_{M3}(l) \quad (10)$$

Similarly to the case of correlation between Hough spectra, multiple local maxima may emerge when computing the cross correlation between *X-spectra*. Each of the maxima is associated with a candidate translation to align the two maps and can be individually tracked.

3.4.2 Map Merge Validation

We have performed an intensive test of the Map Merge block, in order to properly set the parameters of the algorithm proposed in [6] and to verify the effectiveness and speed of the merging algorithm.
To perform the tests we have considered two different parts of the map of Figure 7 (used also in the simulation tests). These submaps are shown in Figure 8 (a) and (b).

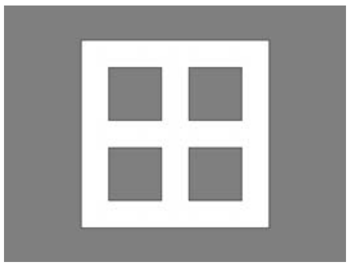


Figure 7: The map used to validate the Map Merge block, used also in the simulation tests.

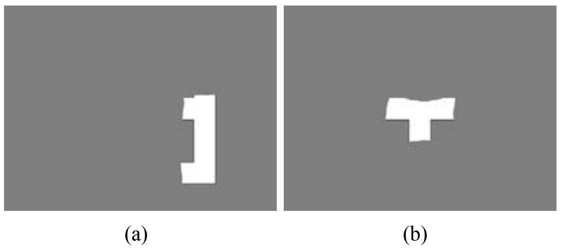


Figure 8: The two parts of map in Figure 7 used to validate the Map Merge block.

For each submap we have generated every possible x translation in the range [-10;10] pixel with resolution 1 pixel, for every x translation every possible y translation in the range [-10;10] pixel with resolution 1 pixel, and for every y translation every possible rotation in the range [0;360] degrees with resolution 1 degree. To evaluate the performances of the merging procedure, given two maps $M1$ and $M2$ we consider the following simple acceptance index:

$$w(M1,M2) = \begin{cases} 0 & \text{if agr} = 0 \\ \frac{agr(M1,M2)}{agr(M1,M2)+dis(M1,M2)} & \text{if agr} \ne 0 \end{cases} \quad (11)$$

where $agr(M1,M2)$ is the *agreement* between $M1$ and $M2$, the number of cells in $M1$ and $M2$ that are both free or both occupied and $0 \le w \le 1$. The *disagreement* between $M1$ and $M2$ (indicated by dis($M1$,$M2$)) is the number of cells such that $M1$ is free and $M2$ is occupied and vice-versa. Notice that only free or occupied cells are considered, while unknown cells are ignored.
We have considered four possible merging algorithms. The first one (MA1) is the one presented in Section 3.4.1, while the other two are the "robust" versions of the first one.
The second one (MA2) evaluates other candidate solutions rotating the ones computed by MA1 in the range [-0.5;0.5] degrees with a step of 0.25 degrees. The third one (MA3) evaluates other candidate solutions rotating again the ones

obtained by MA1 in the range [-1;1] degrees with a step of 0.25 degrees. In the fourth one (MA4) a further translation is added. The other candidates solutions are determined rotating the MA1 solution in the range [-0.25;0.25] degrees with a step of 0.25 degrees, and adding to each rotation a translation of +1 and -1 pixel.
The results in terms of acceptance and computation time of the four merging algorithms are indicated in Table 1.

Methods	Acceptance	Computation Time [s]
MA1	0.9977	0.67
MA2	0.998	1.38
MA3	0.998	1.99
MA4	0.998	2.01

Table 1: Results of the four methods.

For each method we averaged all the acceptance indexes obtained for each rotation (and also translation for M4). The computation time for each method has been computed as the average among all the single computation times for each rotation/translation. The Map Merge validation has been performed on a 2.5 GHz Intel© Core2 Duo platform with 2 GB of RAM memory.
After the evaluation of the accuracy of the four proposed methods and their computation time, we chose to use the MA1 method in the Map Merge block, because the acceptance obtained is comparable with the other methods, and the computation time is lower than the others.

4 Simulation Tests

We consider a simulated environment of a logistic area (see Figure 7). The occupied black areas can be thought as containers or similar bulky items stored before distribution. The dimension of the whole environment is 35×35 m, the black areas are 10×10 m and the corridors are 5 m wide.
We assume that, when the rover is correctly localized, a virtual fork-lift removes or adds one container every minute.
To demonstrate the effectiveness of the proposed approach we first provide results related to $nr = 10$ averaged runs, and the Δ-mapping updating process lasts for approximately two hours each run.
We define the localization error of the robot as the distance between the ground-truth Cartesian position $(x_i^{gt}(t), y_i^{gt}(t))$ and its Cartesian position estimation given by (1) as

$$e^{\rho}(t) = \sqrt{(x^{gt}(t) - \hat{x}(t))^2 + (y^{gt}(t) - \hat{y}(t))^2}. \quad (12)$$

We then define the average localization error over nr runs as

$$\bar{e}^{\rho}_{nr}(t) = \sum_{i=1}^{nr} \frac{e^{\rho}(t)}{nr}. \quad (13)$$

The localization error is shown in Figure 9. The localization error remains lower than 0.5 m, except for a period of time where the error increases. This is due to the loss of the localization by the rover in one of the runs. However the robot quickly recovers correct localization.

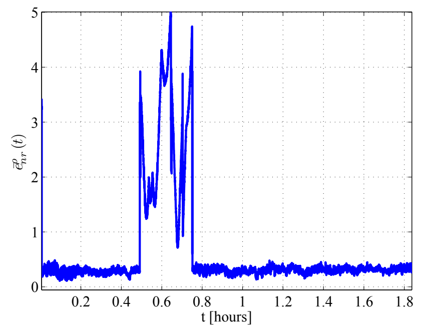


Figure 9: Localization error of a robot in nr runs.

Table 2 shows the acceptance index (11) mediate over the single run, along with the number of variations occurred detected per run.
The number of detected variations in each run depends on the path followed by the robot. In this work the robot wanders in the environment, and there is no active mechanism to ensure the detection of *all* the variations occurred in the environment. Future works will be devoted to enhance the motion strategy. The average quality of the map is comparable with the quality of a map obtained by a Rao-Blackwellized SLAM process in static conditions, since in that case the value of the index obtained is 0.98.

Run	Acceptance	Number of variations
R1	0.979	23
R2	0.98	30
R3	0.9695	68
R4	0.9758	77
R5	0.9728	97
R6	0.9784	47
R7	0.9804	62
R8	0.9731	200
R9	0.9681	197
R10	0.97	100

Table 2: Acceptance values and number of variations of the nr runs.

In another test, we have performed a Δ-mapping process lasting for approximately seven hours, for a total number of 420 variations, to check the long operativity performance. Figure 10 shows the status of the grid map respectively after few variations and after many variations. The map shown in Figure 10 (c) contains open squares because sometimes the robot may not be able to completely map a variation. Moreover, the a-priori knowledge of the geometric aspect of the elements inside the environment (e.g., the goods shape) is not used to complete the final map obtained by the proposed procedure.
In Figure 11 the localization error of the simulated rover is shown. Figure 12 shows the acceptance index (in the range 0-1 from worst to best) related to the map quality over time. After seven hours of operation the rover localization error remains acceptable (below 1 m) and the quality of the map is still comparable with the quality of a map obtained by a Rao-Blackwellized SLAM process.

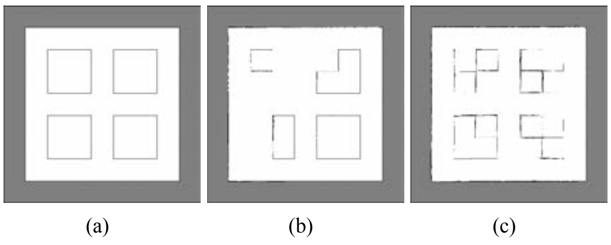


Figure 10: The initial map (a), the map after few variations (b), and the map at the endo of the Δ-mapping process.

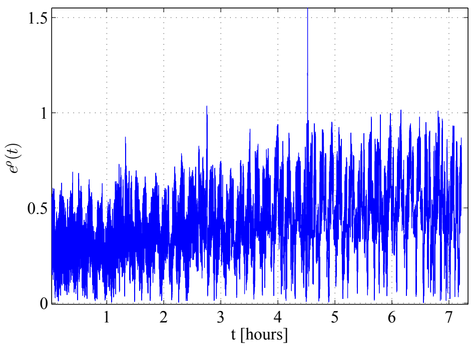


Figure 11: The localization error in the long operativity test.

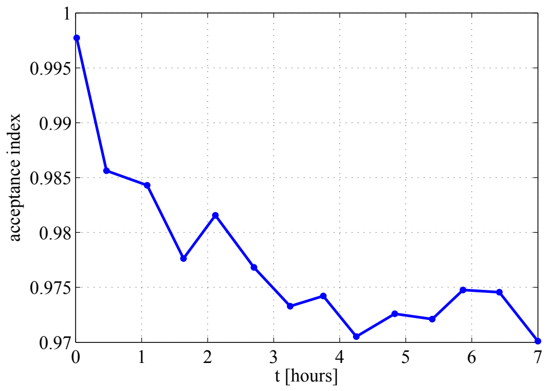


Figure 12: The trend of the quality of the map over time.

5 Experimental Tests

We have also performed some experiments in a real environment using a Pioneer P3DX robot endowed with a SICK LMS200 laser rangefinder and connected with an Acer AspireOne with 512 MB of RAM memory and powered by an Intel Atom 1.6 GHz. To perform the experiments, first we have placed some simulated goods in a 10×4 m hall, each obstacle being 0.6×1.2 m and made of 6 boxes of 0.3×0.4 m and we have performed a classical Rao-Blackwellized SLAM process to obtain the map of the environment.
Then, to test our Δ-mapping approach, we have first removed 4 boxes from one of the obstacles and placed them again in the same position as before, letting the robot perform Δ-mapping phases.
The map produced by the SLAM process is shown in Figure 5 (a) along with a snapshot of the environment in (b), acquired while the SLAM process was performed. In (c) and (e) of the same Figure the two maps resulting from two Δ-mapping processes are shown, along with the status of the obstacles in (d) and (f).
By visually inspecting the maps shown in Figure 5 (c) and (e) we can notice the effectiveness of the Δ-mapping phases, because the presence and absence obstacles has been mapped, and the quality of the maps obtained is satisfactory.

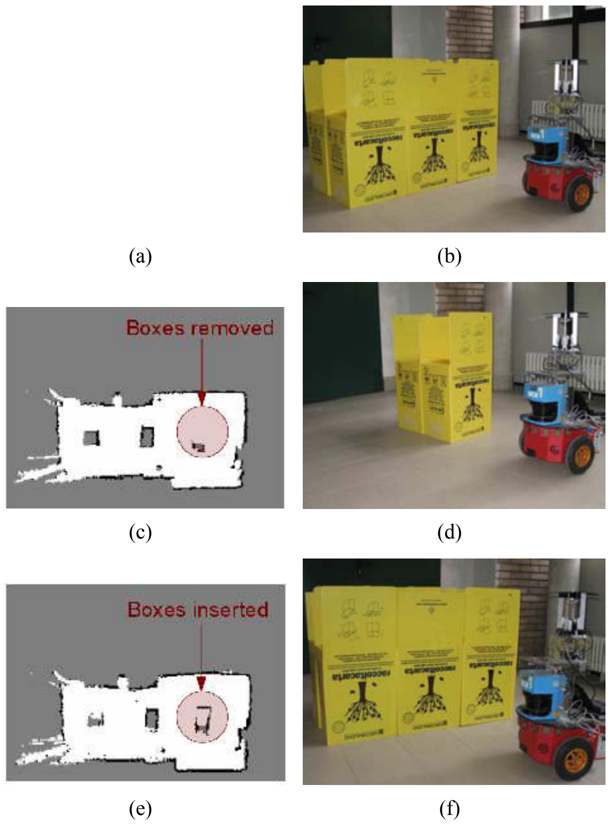


Figure 13: State of the map and snapshots of the environment in the real test.

The proposed approach is also light from a computational point of view (especially if compared to a particle filter based SLAM approach), because it limits the demand for computational resource only to the Δ-mapping phases. In Figure 14 the peaks of cpu usage are visible in the upper plot, while the peaks of memory occupation are shown in the lower plot. In particular the memory required by the application does not exceed 92 MB, and only for a short period of time. The lightness of the approach enables its use in those applications where other services have to be carried out in parallel with the Δ-mapping process.

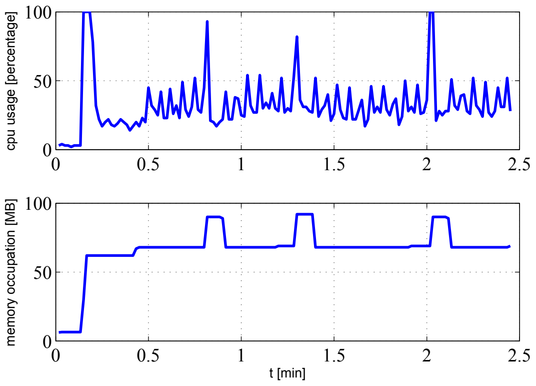


Figure 14: Cpu usage (upper plot) and memory occupation (lower plot) in the real experiment.

6 Conclusions

In this paper we have proposed a methodology called Δ-mapping, which is able to perform map updating from an initial map when dynamical variations occur in the environment. This methodology enables robots to detect variations, to generate a local map containing only the persistent variations, and finally to merge the local map with the one used for localization. The variations in the environment are detected using a technique called *weighted recency averaging*, while the local maps are merged employing a technique based on the Hough transform. The approach is suitable for applications such as logistic applications, where a long-term operativity is required and the algorithm performing the map update has to be computationally light and to use limited memory, to allow concurrent execution of other services. Future works will be devoted to the extention of the Δ-mapping process to a multirobot team, and to develop team coordination strategies that actively search modifications in the map.

References

[1] Rmap. Website. http://www-robotics.usc.edu/ ahoward/pmap/rmap_8h.html.

[2] F. Abrate, B. Bona, M. Indri, S. Rosa, and F. Tibaldi. Switching multirobot collaborative localization in symmetrical environments. In *IEEE International Conference on Intelligent RObots Systems (IROS 2008), 2nd Workshop on Planning, Perception and Navigation for Intelligent Vehicles (PPNIV)*, 2008.

[3] F. Abrate, B. Bona, M. Indri, S. Rosa, and F. Tibaldi. Three state multirobot collaborative localization in symmetrical environments. In *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, pages 1–6, 7th May, 2009.

[4] T. Bailey, J. Nieto, J. Guivant, M. Stevens, and E. Nebot. Consistency of the ekf-slam algorithm. pages 3562 –3568, oct. 2006.

[5] Peter Biber and Tom Duckett. Dynamic maps for long-term operation of mobile service robots. In *In Proc. of Robotics: Science and Systems (RSS*, 2005.

[6] Stefano Carpin. Fast and accurate map merging for multi-robot systems. *Auton. Robots*, 25(3):305–316, 2008.

[7] G. Grisetti, C. Stachniss, and W. Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *Robotics, IEEE Transactions on*, 23(1):34–46, Feb. 2007.

[8] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, 4:333–349, 1997.

[9] Adam Milstein. Dynamic maps in monte carlo localization. In *Canadian Conference on AI*, pages 1–12, 2005.

[10] Luis Moreno, Santiago Garrido, Dolores Blanco, and M. Luisa Munoz. Differential evolution solution to the slam problem. *Robotics and Autonomous Systems*, 57(4):441 – 450, 2009.

[11] L.M. Paz, J.D. Tardos, and J. Neira. Divide and conquer: Ekf slam in $o(n)$. *Robotics, IEEE Transactions on*, 24(5):1107–1120, Oct. 2008.

[12] Y. Rachlin, J.M. Dolan, and P. Khosla. Efficient mapping through exploitation of spatial dependencies. In *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 3117–3122, Aug. 2005.

[13] S. Thrun, W.Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2005.

[14] N. A. Vlassis, G. Papakonstantinou, and P. Tsanakas. Dynamic sensory probabilistic maps for mobile robot localization. In *In Proc. IROS'98, IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 718–723, 1998.