Temporal Occupancy Grid for mobile robot dynamic environment mapping

Nikos C. Mitsou, and Costas S. Tzafestas
School of Electrical and Computer Engineering
Division of Signal, Control and Robotics
National Technical Univ. of Athens
Zografou Campus, Athens 15773, Greece
Email: nmitsou@mail.ntua.gr, ktzaf@softlab.ntua.gr

***Abstract*— Mapping dynamic environments is an open issue in the field of robotics. In this paper, we extend the well known Occupancy Grid structure to address the problem of generating valid maps for dynamic indoor environments. We propose a spatiotemporal access method to store all sensor values (instead of preserving only one value for each cell as in the common occupancy grid case). By searching for similar time series, we can detect moving objects that appear only in a limited number of possible configurations (e.g. doors or chairs). Simulated experiments demonstrate the potentialities of the proposed system.**

I. Introduction

Robotic mapping is regarded as one of the most important problems in the field of robotics. Truly autonomous robots must be able to create an accurate map of their environment. During the past few years, many algorithms have emerged that produce excellent maps of unknown environments in real time [9], [12].

However, most mapping algorithms are based on the assumption that the environment is static. In the case of dynamic environments, these algorithms are most likely to fail. Places with high dynamism (e.g. doors or passages) cannot be classified correctly as free or occupied areas. The most common approach to deal with the dynamic effects is to consider them as noise and thus remove them from the map ([8], [7]). By applying this filtering, however, we ignore important information of the environmental structure. The most important consequence of this is that localization will be less accurate when the robot is examining these areas as sensor measurements will not match with the removed features. Therefore, mapping algorithms that can cope with the dynamic nature of the environment are of great importance.

In this paper, we propose a new mapping technique, the Temporal Occupancy Grid (TempOG) algorithm. The TempOG algorithm is capable of mapping environments where objects change places over time. We classify objects in the robot environment into three categories: a) Static objects, objects that do not change their position over time (e.g. walls, beds or locked doors) b) Low-dynamics objects, objects that appear in a specific number of places (e.g. chairs or doors) and c) High- dynamics object, objects that move arbitrarily in the environment and can be found in many different positions (e.g. humans). The proposed algorithm can correctly identify and model static, Low- dynamics and High- dynamics objects.

The basic idea behind TempOG is that we extend the typical occupancy grid [13] through the time dimension. We assign a time index (a special B+ tree used to index time intervals) to every cell of the grid and we store all occupancy probabilities of the cell into the index. In this way, we can correctly conclude on the state of every cell. To extract moving objects from the environment (e.g. doors, chairs or any areas that change over time), we consider time index leaf nodes as time series and we search for similarities in subsequent time-series. In this work, we assume that robot odometry measurements are perfect, so robot position is known at any time and no localization is needed.

This paper is organized as follows: In Section II, we present a brief survey of related work in the area of mapping dynamic environments. In Section III, we present the proposed algorithm and in Section IV we describe what kind of information can be extracted from the implemented structure. Section V describes the simulations conducted and presents the experimental results obtained. Finally, conclusive remarks and future work directions are given in Section VI.

II. Related Work

In the last few years, many attempts have been made towards mapping of dynamic environments. Two lines of research have emerged: a) detection and modeling of moving people and b) identification of different environmental configurations.

There exist many approaches that deal with the first line. For example, in [11] a sonar range sensor, a camera and differentiating techniques were used, in [10] an expectation maximization algorithm was applied and in [17] a probabilistic filtering algorithm was implemented all to detect moving people in the robot environment.

For the second line, which is the subject of this paper, the authors of [1] implemented a map differencing technique to find objects that move slowly in the robot environment. Models of non-stationary objects were learned with the use of an expectation maximization algorithm from a sequence of occupancy grid maps captured in different points in time. However, the moving objects must be uniquely identified by their shape, otherwise wrong associations might be learned.

In [4], the authors use particle filters and conditional binary Bayes filters to estimate the state of doors in the environment. In their experiments, binary state objects were successfully detected in a predefined environment.

In [2], an expectation maximization algorithm was also used to detect and model doors. Laser and camera readings were used to identify walls, moving and non-moving doors.

In [18], a fuzzy clustering technique was presented to capture the typical configurations of the dynamic environment. Sub-maps were collected in different points in time and were grouped into possible configurations. The environment was assumed to be static during the collection of the sub-maps.

In [19], two occupancy grids were maintained, the first was used to model the static parts of the environment while the second to store the dynamic objects. However, no different configurations of the environment were detected.

In [3], the authors follow a similar idea by extending the Occupancy Grid algorithm through the time dimension. In their work, three or more Temporal Occupancy Grids with different sample rates (timescales) were preserved in order to conclude on the state of every cell. However, in order to classify cells as occupied a simple aggregation is performed not exploiting in full extent the information contained in the stored data.

In the following sections, we will presented the proposed algorithm. Contrary to the previous works, the algorithm in this paper can identify all different configurations of an unknown environment with the use of a laser range finder and without the assumption that the environment will be static during the experiments or that the moving objects will be uniquely identified by their shape.

III. Temporal Occupancy Grid (TempOG)

Intuitively, an ideal solution to the mapping problem would exploit the whole history of the robot perception of the environment. Following this intuition, we preserve in an efficient way all robot stimulus of the environment and utilize the information for mapping of non-stationary environments. Contrary to existing approaches that store aggregate values per cell, we store the whole history of evolution; in this way, we can extract much more information of the environment (e.g. detection of different environmental configurations).

To achieve this, we extend the Occupancy Grid across the time axis, resulting in a new structure, the Temporal Occupancy Grid (TempOG). TempOG stores the history of the occupancy probability for each cell through an appropriate index structure, the Time Index [5]. Thus an environment of $n \times n$ cells, contains $n \times n$ Time Index structure to form a forest of Time Indexes.

A short introduction to the Time Index follows.

A. Time Index

Time Index is a special case of B+ tree [14] [1] that is used for storage and retrieval of values that are valid during specific time periods. A Time Index of order n is a search tree with the following properties:

- The root has at least two sub–trees unless it is a leaf and at most n children.
- Each non-root and non-leaf node holds k pointers to sub trees and $k-1$ separators ($k \geq \lceil n/2 \rceil$ and $k \leq n$)
- Key values appear in the leaf nodes.
- Leaf nodes are at the same level.
- Leaf nodes are linked.

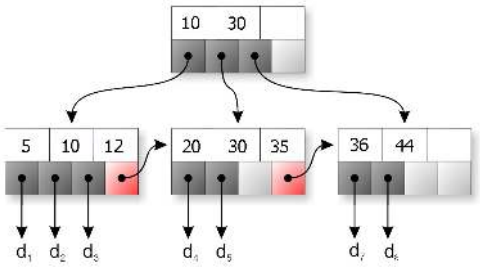


Fig. 1. An example of a Time Index of order 3

The Time Index was specifically designed for indexing temporal data. The time dimension is represented using the concept of time intervals, as in [5]. A *Time interval* $[t_i, t_j]$ is a set of consecutive (equidistant or not) time points, where t_i is the first point and t_j is the last one. A single time point t can be represented as $[t, t]$.

The Time Index differentiates from the B+ Tree in the fact that due to the monotonic nature of time, deletions never occur while updates occur in an append mode. So, new entries will always be inserted in the rightmost node of the tree and the complexity of the insertion will always be O(1). When the rightmost node is full, a new node is created and the changes propagate upwards, just as in B+ trees[2]. (An extensive comparison of available time indexes can be found in [16]).

An example of a Time Index is shown in Figure 1. 8 time intervals exist in this tree: $[0, 5]$, $[5, 10]$, $[10, 12]$, $[12, 20]$, $[20, 30]$, $[30, 35]$, $[35, 36]$ and $[36, 44]$. Each i th interval points to value d_i, except for the sixth interval ($[30, 35]$) which points to value *null*.

B. Temporal Occupancy Grid – $TempOG$

As we have already mentioned, we use a $n \times n$ grid and an underlying forest of $n \times n$ Time Indexes (one for each cell of the grid). Each index stores the probability of the corresponding cell being occupied at a given time interval.

Example

Consider the example of Figure 2, where the robot is standing in front of a door. The door is partitioned into three grid cells and it is closed during the interval $[0, t_1)$, open during $[t_1, t_2)$ and closed during $[t_2, now)$. In this

[1] B+tree is a widely used index structure in the database domain

[2] In a simpler case, where we are not interested in efficiently traversing the occupancy history, we can use a simple linked list instead of a Time Index to avoid the rebalancing cost.

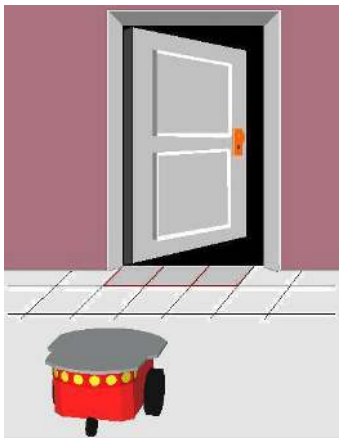


Fig. 2. TempOG: A robot in front of a door. The closed door configuration is split into three cells.

example, we assume that the door is a binary state object and the cells can take either the free or the occupied value.

Initially, the Time Indexes of the three cells are empty. At time point 0 when the door is closed, the sensor values will indicate that the three cells are occupied. A time interval will be created in the leaf nodes of the three Time Indexes that will store the occupied value with start point 0 and stop point 0. At the next time point (time point 1), the occupied value will still be detected and the previous time interval will be expanded to end at time point 1. When the time point t_1 is reached, this time interval will end at t_1. At this time point, the door opens and the cells are detected to be free. So a new time interval will be created that will hold the free value. In the end, the Time Indexes of the three cells that correspond to the door will be storing the values of Figure 3 a).

In real robot environments, the probability of a cell being occupied can take any value from zero to one. However, due to sensor measurement noise, subsequent measurements of an obstacle could yield in slightly different occupancy values. To avoid creating time intervals for every slightly different value, we use a threshold to group similar values. For example, if the current time interval stores the value 0.5 and a new value of 0.51 arrives, no new time interval will be created, but the already created interval will be used to store the value and its end time point will be updated. The value of the threshold is important: If a small value is selected, more time intervals will be created requiring more processing resources. On the other hand, if a big value is selected, important features of the environment might be lost.

Another important issue is the fact that only a few cells are related to every sensor measurement. Therefore, the cells occupancy is not known during some time intervals *(gaping effect)*. This means that leaf nodes of the underlying Time Index can point to $null$ values. In the above example, for instance, imagine the case, at time point t_1, where a human stands between the robot and the door. At this point, the robot cannot observe the state of the door cells. If at t_2 the human exits the scene, the state of the door cells will be again known and the Time Index

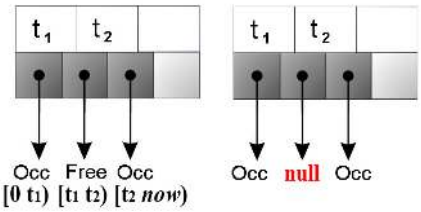


Fig. 3. TempOG: a) A simple example. b) The gaping effect.

will be storing the values of Figure 3.

The algorithm for grid construction is presented in Figure 4. When a new sensor measurement arrives, the occupancy probabilities of the affected cells are calculated and propagated to the corresponding Time Indexes; depending on the incoming values, the latest time intervals are updated or new time intervals are created.

Build_Grid()

```
FOR each cell c_i BEGIN
    Initialize_Time_Index (TI(c_i));
END

FOR each measurement v_i referring to cell c_i BEGIN
    interval = Get_Last_Time_Interval(TI(c_i));
    IF (now − interval.stop) > thres_1 THEN BEGIN
        //gaping effect
        Create_New_Time_Interval(TI(c_i),v_i,now)
    END
    ELSE BEGIN
        IF (|v_i - interval.value| > thres_2)
        THEN BEGIN
            //gaping effect
            Create_New_Time_Interval(TI(c_i),v_i,now)
        END
        ELSE BEGIN
            interval.stop = now;
        END
    END
END
```

Fig. 4. Grid Building Algorithm

IV. Extracting Information from the Temporal Occupancy Grid

TempOG efficiently stores the occupancy history of each cell of the environment. This wealth of information can be exploited for extracting valuable information regarding the robot environment.

In the following paragraphs, we present some representative applications of the TempOG structure.

A. Generating map dynamics

By using TempOG, we can easily generate the map dynamics. This map indicates how fast the occupancy of the areas/ cells changes over time. As a metric for the cell's dynamics, we use the standard deviation of the probabilities of the Time Index of the cell. A small standard deviation indicates that the values in the Time

Index are close to each other and the cell is static. On the other hand, a big standard deviation indicates great differences in the values of the Time Index and, thus, a dynamic cell.

This algorithm is presented in Figure 5.

Get_Cell_Dynamics(cell c)

```
//calculate standard deviation
mean = Get_Mean_Value(TI(c));
For each value in TI(c)
    v =v + | value - mean |

return v / get_Length_of_Time_Index(TI(c));
```

Fig. 5. Dynamic cells

B. Generating static map

By *static objects* we refer to objects that do not change their position over time (e.g. walls, beds, locked doors, etc.). The algorithm presented in the previous paragraph can be also used to detect the static objects of the environment. Cells with low dynamics and a high probability of being occupied are considered as static.

A pseudo-algorithm for deciding whether a cell is static is shown in Figure 6.

Is_Static(cell c)

```
IF (Get_Cell_Dynamics(c) < thres1) AND
    (Get_Mean_Value(c) > thres2)
    return STATIC
ELSE
    return NON_STATIC
```

Fig. 6. Static cells

C. Detecting Dynamic Objects

Dynamic objects are objects whose position changes over time. We can distinguish them into two categories:

- *Low Dynamic objects:* are objects that appear in a specific number of places. Chairs and doors belong to this category.
- *High Dynamic objects:* are objects that move arbitrarily in the environment and can be placed in many different positions. Humans belong to this category.

In the following paragraphs, we present how dynamic objects can be detected.

1) Detecting Low Dynamic Objects: As already stated, low dynamic objects are moving objects that can be positioned in a number of different configurations. The most common example is a door that can be either open or closed. The detection of low dynamic objects requires the ability to detect all possible object configurations.

With TempOG, we can detect Low Dynamic Objects without any prior knowledge of the object size or motion. The detection of Low Dynamic Objects is carried out in two steps:

- Step 1: Find possible object configurations by grouping neighboring areas that follow the same motif.

Associate_Object_Configurations()

```
FOR each cell c in cells C BEGIN
    N = Get_Neighboring_Cells(c);
    FOR each n ∈ N BEGIN
        IF (distance(Time_Series(c),Time_Series(n))¡τ)
        THEN
            Associate_Cells(c,n);
    END
END
```

Fig. 7. Detect object configurations

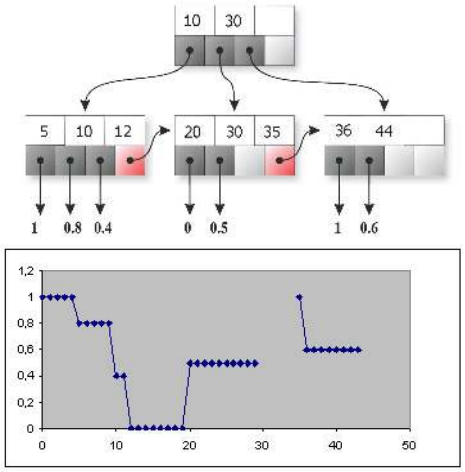


Fig. 8. The Time Index and the corresponding Time Series

- Step 2: Find configurations that belong to the same moving object.

Below, we provide more detail on each of these steps:

a) Step 1: Find possible object configurations: A low dynamic object can be considered to cover more than one cell in the environment (a door, for example, might cover three or more cells). Such an object falls into different configurations/ states (in the previous example, the states could be $open$ and $closed$). In order to detect these configurations, we search through the history of all cells to find neighboring cells that change with the same motif. These cells correspond to one of the possible configurations of the object. For example, in the case of a door, all cells of the closed door configuration would have the same values, regardless of the door's state.

To find cells that change with the same motif, we treat the values in the leaf nodes of a Time Index as a Time Series (a collection of observations made sequentially in time) that describes the cell occupancy. An example of a Time Index for a specific grid cell and its equivalent time series is presented in Figure 8.

A single time series describes the evolution of the occupancy of the corresponding cell over time. Similar time series indicate similar cell occupancies. Thus, in order to find a single configuration of a moving object, we aim in finding neighboring cells with similar time series. An appropriate algorithm is shown in Figure algo:findConf.

There exist various similarity measures in the data

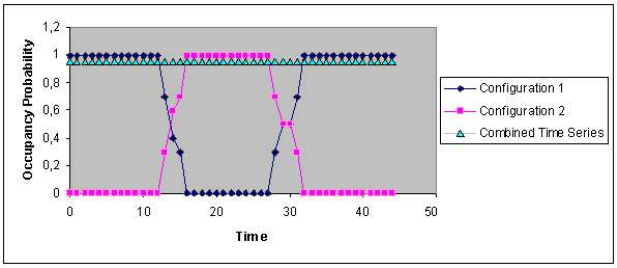


Fig. 9. The Time Series of the two configurations and the combined Time Series

Find_Two_Configuration_Objects

```
FOR each p1 in patterns BEGIN
    FOR each p2 in patterns BEGIN
        combination = p1.getTimeSerie() +
                      p2.getTimeSerie();
        IF (combination.isValidObject()) THEN
            p1.associate(p2);
    END
END

}
```

Fig. 10. Find two-state objects

mining community. In this work, we use the Manhattan distance, a special case ($p = 1$) of the Minkowski distance [15]:

$$D_{Minkowski}(T_1, T_2) = \sqrt[p]{\sum_{i=1}^{n}(t_{1i} - t_{2i})^p}$$

Any other distance measure could be applicable.

When similarity of two time series is above a given threshold, the corresponding cells are assumed to be covered by the same configuration.

b) Step 2: Associate configurations: In this step, we search for correlations among the patterns found in the previous step. We search for patterns that represent different configurations of the same object. The simpler case is the case of two-state objects, i.e. objects that can appear in two different positions. In such objects, their configurations are complementary; when the one is occupied the other one is free. Thus, in order for two different patterns to belong to the same object, their combined Time Series must contain the occupied value at any time, as shown in Figure 9.

The above method is too restrictive. Consider the case where a human passes through an open door. At this time, the open door cells will appear as occupied and the combined Time Series will be invalid (value 2 ¿ 1 for some time points). Also, the robot might not observe both the patterns at the same time points (gaping effect). In this case, the unobserved areas of the Time Series are filtered out during the evaluation.

The pseudo-code of the algorithm can be found in Figure 10.

Following the same rationale, we can search for objects with three, four, five etc different configurations. The difference is that we have to combine and evaluate more than two Time Series.

Find_Highly_Dynamic_Objects()

```
FOR each occupied cell c in measurement M BEGIN
    IF Is_Static(c) == NON_STATIC THEN
        IF NOT Part_Of_Configuration(c) THEN
            Is_High_Dynamic_Object(c)
    END
END
```

Fig. 11. Detect High Dynamic object

In order for the algorithm to correctly associate the different patterns, the moving objects must change their positions with different time rates. If, for example, two doors follow the same motion pattern, the algorithm will be confused in the pattern association step.

2) Detecting High Dynamic Objects: We can detect High Dynamic Objects as follows: We extract the occupied cells indicated by the latest sensor measurement. Those cells that are not static and do not belong to a Low Dynamic Object configuration are automatically identified as part of a High Dynamic object. The pseudo algorithm is presented in Figure 11.

Although we preserve the occupancy history of the environment, with the above algorithm we cannot track High Dynamic Objects or extract their trajectory. The objective of the algorithm is to generate the Highly Dynamic map of the environment.

D. Focused TempOG Exploitation

So far, the applications of TempOG involve the whole history of observation. In practice however, a user might be interested in retrieving the history for a specific period of interest described through a time interval of the form $[t_{start}, t_{end}]$. Adjusting the above described algorithms to deal with the focused environmental monitoring is straightforward.

V. Experiments

In order to validate the proposed structure, extensive simulated tests have been performed. In our experiments, a simulated ActiveMedia Pioneer II Robot was equipped with a laser range finder. The goal of the experiments was to present the effectiveness of our method in creating a valid map of the environment. Furthermore, we were interested in evaluating the algorithm's ability to detect doors that have changed their state during the experiment.

For each experiment, we present the occupancy grid map as it would be generated by the common occupancy grid algorithm and the map as generated by the proposed algorithm. In this map, we present the static areas of the environment as black areas and the unexplored areas as gray. Also, we present all the patterns that were found in the environment with different colors. Patterns found to represent the same object are drawn with the same color and are connected with a line.

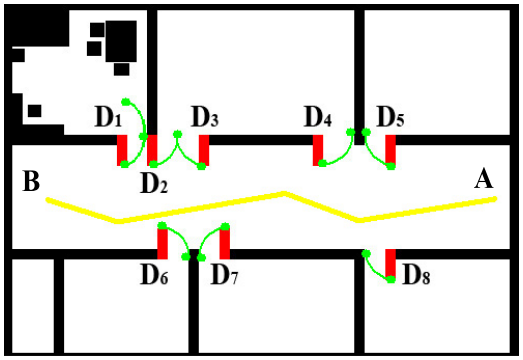


Fig. 12. Robot environment. The yellow line represents the robot path (from A to B and back to A)

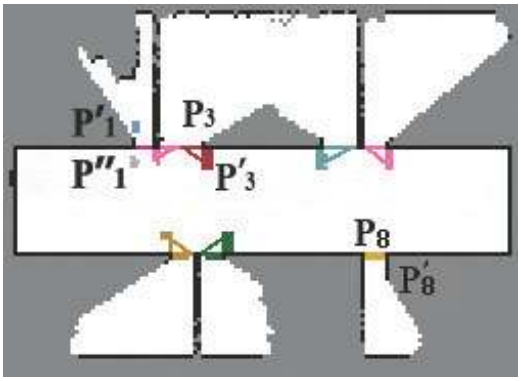


Fig. 13. Generated Map.

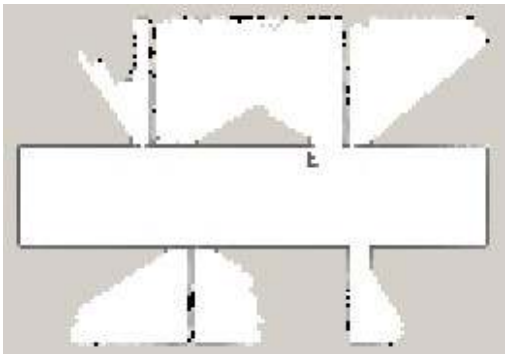


Fig. 14. Occupancy Grid Map

A. Implementation Issues

The main disadvantage of the proposed method is the fact that great amounts of memory might be required by the spatiotemporal structure to monitor a highly dynamic environment during a long period of time. We propose the following three solutions that can decreased the amount of memory needed:

- We can compress the time series [6].
- We can define a time window outside of which values are store in the disk or deleted for ever.
- We can replace portions of time series as references to objects.

B. Experiment on non - crowded environment

The first experiment has been carried out in a long hallway. The simulated environment used in this experiment can be found in Figure 12. Red areas are doors that change their state during the robot exploration. All doors have two possible states (open and closed) except for door D_1 which has three possible states (one closed and two different open states).

In Figure 13, the robot has created a correct map of the static areas of the environment. Also, all the doors have been identified correctly as moving objects but only six of them have their configurations been assigned to them. For example, the door D_3 has been identified to correspond to patterns P_3 and $\acute{P}_3$. On the other hand, although all three configurations of door D_1 were detected, they could not be assigned as one object. The reason for this is that the $\acute{P}_1$ configuration could not been monitored enough, so the corresponding time series is not adequate to assignment.

In the case of door D_8, only one configuration could be detected (P_8). The other configuration ($\acute{P}_8$) was mistakenly taken for static object. This happened because every time this area was observed (door open), it was sensed as occupied area. Whenever this area was unoccupied, the robot could not observe it because the door was closed.

In Figure 14, the typical occupancy grid map of the environment is presented. We can observe that only one configuration for each door is drowned (either the closed or the open configuration). Also, in the cases of the D_1, D_4 and D_5 doors, not all points of their configurations were detected.

C. Experiment on crowded environment

In order to evaluate the effectiveness of the algorithm under tougher conditions, we used a more challenging scenario. We added a number of moving people inside the robot environment. Three different simulations are presented: Experiment with 2.a) 20 people, 2.b) with 40 people and 2.c) with 60 people. Different people behaviors were implemented with regard to the motion speed and the changing direction rate.

The results can be found in Table I. We can see that although the environment was extremely dynamic, the extracted map is of high quality. Static areas of the environment are correctly identified. Almost all different configurations of low dynamics objects are detected. The generated map of Experiment 2.a) is almost identical to the map of Experiment 1. In experiment 2.b), the robot mistaken an area close to a door as a possible door configuration. In the last simulation, more associations between the different configurations are corrupted. The main reason to these corruptions is the fact that robot visibility is reduced due to the moving people. The doors are not frequently observed and moving people can be mistakenly recognized as Low Dynamics object configurations.

Another issue is that unrealistic configurations are detected. Areas that do not belong to any Low Dynamic object are identified as new possible configurations. However, these configurations can be related to people that move slowly in the environment. These people can be

considered as Low Dynamic objects and thus be modeled by the algorithm.

VI. Conclusions

In this paper, we presented a spatiotemporal structure to model and generate maps of dynamic environments. Our approach uses a forest of Time Indexes to preserve all occupancy probabilities for each cell. With this structure, we showed that many important dynamic features of the environment are preserved and not filtered out as in many existing algorithms. By traversing the leaf nodes of the Time Indexes, we can extract information such as the static objects map, the dynamic and static areas of the environment and the moving objects. Simulated experiments have indicated the potentialities of the algorithm.

An important advantage of the proposed structure is the fact that all the algorithms that use Occupancy Grids can be easily modified to work with the proposed structure with no further adjustments. For example, existing algorithms for detection and tracking of moving objects (people, robots) and algorithms for localization on dynamic environments can be applied as is and are expected to generate the same results.

However, our goal for the future is to search for algorithms that will exploit the spatiotemporal nature of the proposed structure. It would be interesting to extend existing algorithms for dynamic environment localization and for people detection and tracking or create new ones that will detect patterns in time and take advantage of this additional knowledge. Another goal is to test TempOG on a real robot and examine the results. An issue that will most probably arise is the need for localization due to the odometry errors that unavoidably come with the robot motion. So, a SLAM algorithm will be investigated.

Acknowledgment

This work was partially co-funded by the European Commission and the Hellenic General Secretariat for Research and Technology (GSRT) under Measure 3.3 of the Operational Programme "Information Society" in the 3^{rd} Community Support Framework (National Project Name: DIANOEMA, ID: 35).

References

[1] D. Anguelov, R. Biswas, D. Koller, B. Limketkai, S. Sanner, and S. Thrun. Learning hierarchical object maps of non-stationary environments with mobile robots. In *Proc. of the 17th Annual Conference on Uncertainty in AI (UAI)*, 2002.

[2] D. Anguelov, D. Koller, E. Parker, and S. Thrun. Detecting and modeling doors with mobile robots. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2004.

[3] D. Arbuckle, A. Howard, and M. J. Matarić. Temporal occupancy grids: a method for classifying spatio-temporal properties of the environment. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 409–414, Lausanne, Switzerland, Oct 2002.

[4] D. Avots, E. Lim, R. Thibaux, and S. Thrun. A probabilistic technique for simultaneous localization and door state estimation with mobile robots in dynamic environments. In *Proc. of the Conference on Intelligent Robots and Systems (IROS)*, Lausanne, Switzerland, 2002.

[5] R. Elmasri, G. T. Wuu, and Y. J. Kim. The time index: An access structure for temporal data. In *16th VLDB*, pages 1–12, 1990.

[6] E. Fink and K. B. Pratt. Indexing of compressed time series. In *Data Mining in Time Series Databases*. World Scientific, Singapore.

[7] D. Fox, W. Burgard, and S. Thrun. Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, 11:391–427, 1999.

[8] D. Fox, W. Burgard, S. Thrun, and A. Cremers. Position estimation for mobile robots in dynamic environments. In *Proc. of the AAAI Fifteenth National Conference on Artificial Intelligence*, 1998.

[9] J. Gutmann and K. Konolige. Incremental mapping of large cyclic environments. In *Proc. of the IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*, pages 318–325, Monterey, California, November 1999.

[10] D. Hähnel, R. Triebel, W. Burgard, and S. Thrun. Map building with mobile robots in dynamic environments. 2003.

[11] G. Q. Huang, A. B. Rad, and Y. K. Wong. A new solution to map dynamic indoor environments. In *International Journal of Advanced Robotic Systems, Vol. 3*, 2006.

[12] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proc. of the AAAI National Conference on Artificial Intelligence*, Edmonton, Canada, 2002. AAAI.

[13] H. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *Proc. of the IEEE Int. Conf. on Robotics Automation (ICRA)*, 1985.

[14] B. C. Ooi and K. L. Tan. B-trees: Bearing fruits of all kinds. In X. Zhou, editor, *Thirteenth Australasian Database Conference (ADC2002)*, Melbourne, Australia, 2002. ACS.

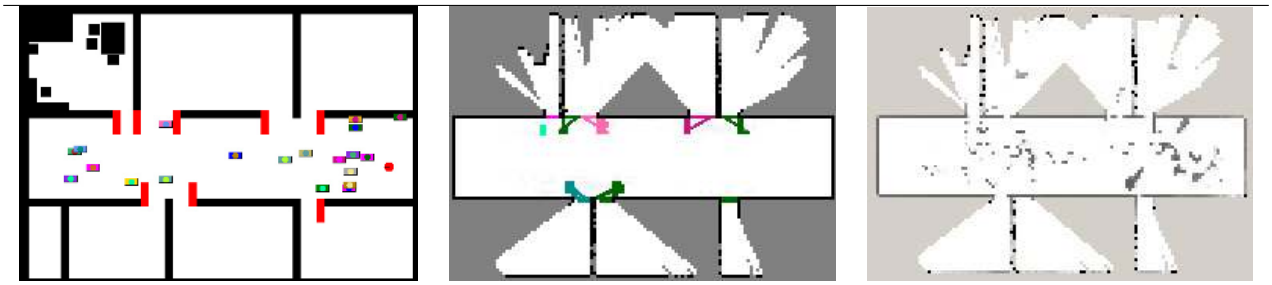
[15] Y. T. Qian, S. Jia, and W. W. Si. Markov model based time series similarity measuring. In *Proc. of the Second International Conference on Machine Learning and Cybernetics*, pages 278–283, 2003.

[16] B. Salzberg and V. Tsotras. Comparison of access methods for time-evolving data. In *ACM Computing Surveys (CSUR)*, 1999.

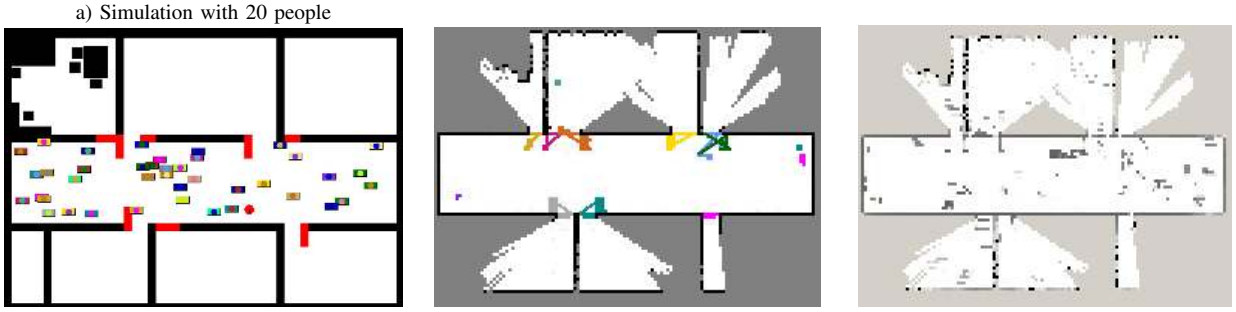
[17] D. Schulz, W. Burgard, D. Fox, and A. Cremers. Tracking multiple moving targets with a mobile robot using particle filters and statistical data association, 2001.

[18] C. Stachniss and W. Burgard. Mobile robot mapping and localization in non-static environments. In *Proc. of the National Conference on Artificial Intelligence*, Pittsburgh, PA, USA, 2005.

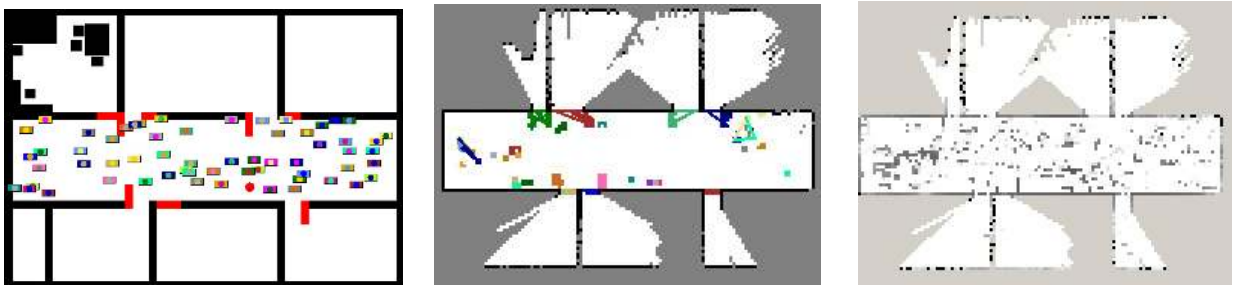
[19] D. F. Wolf and G. S. Sukhatme. Online simultaneous localization and mapping in dynamic environments. In *IEEE International Conference on Robotics and Automation*, pages 1301–1306, 2004.



a) Simulation with 20 people



b) Simulation with 40 people



c) Simulation with 60 people

TABLE I

LEFT IMAGE PRESENTS THE SIMULATED ENVIRONMENT. CENTER IMAGE PRESENTS THE MAP AS GENERATED BY THE TEMPOG ALGORITHM. RIGHT IMAGE PRESENTS THE CLASSIC OCCUPANCY GRID OF THE ENVIRONMENT.