Trail-Map: A Scalable Landmark Data Structure for Biologically Inspired Range-Free Navigation

Annett Stelzer, Elmar Mair and Michael Suppa

Abstract— **Small mobile robots often have very limited computational resources, but should still be able to navigate robustly in spacious unknown environments. While local navigation already requires high-resolution maps for obstacle avoidance and path planning, the global navigation task should aim to consume as little resources as possible but still enable the robot to robustly find the way to important places. Inspired by models of insect navigation, Augustine et al. [1] recently introduced the LT-Map, a scalable data structure for homing based on bearing-only landmark measurements. The robot memorizes landmark configurations during its first traversal of a path and uses them to navigate the same route again. The LT-Map uses a tree structure to store the landmark views in the order of their translation invariance. This paper introduces an improvement of the LT-Map, the Translation Invariance Level Map (Trail-Map). This novel data structure also stores the landmark views in a hierarchical order of translation invariance, but is based on lists of landmark views. Thus, it avoids redundancies that could arise in the LT-Map and leads to a more consistent hierarchy. The Trail-Map achieves significant memory savings and can be created and pruned very efficiently what makes it attractive to mobile robots with limited computational power. Simulation results show that the Trail-Map data structure can save more than 80% of memory compared to the LT-Map while achieving the same path accuracy.**

I. Introduction

A robot operating in an unknown environment must be able to accurately model its close surroundings for obstacle detection and avoidance, as well as for planning and manipulation. High-resolution metric maps are widely used for this local navigation task because they represent geometric structure [2], [3]. Additionally, the robot has to solve the global navigation task, which means it must be able to navigate through the environment, especially to find back to its starting position. In this context, metric maps are also commonly used, either dense grid maps [4], [5] or sparse landmark maps [6], [7]. Metric maps have the advantage that they are human-readable. Dense grid maps can represent the geometry of the environment, but their accuracy depends on the chosen resolution, which is hard to adapt to local environment conditions during runtime. Further, since position estimation errors accumulate over time, computationally expensive algorithms, for example SLAM (Simultaneous Localization and Mapping), have to be used for keeping the global maps consistent. Although much work has been done on making SLAM computationally tractable [7], [8], the computation costs still depend on the size of the map and, hence, will grow unbounded as the map extends.

As a counterpart to metric maps, topological maps model the environment as a graph, where the nodes represent distinct places that are connected by edges. Topological maps represent the connectivity between places very well, are compact representations of the environment, and allow for efficient path planning. Several works use hybrid maps [9], [10], which use local metric maps as nodes and the coordinate transformations between the submaps as edge information. By doing so, the global resolution and, hence, the computational requirements for maintaining the global map, is greatly reduced.

For tasks in spacious environments and for robots with limited resources it is important that the global map can be maintained with low cost and can adapt to the available memory. In particular, planetary rovers have only very restricted computational power, but also other small mobile robots such as micro aerial vehicles (MAVs) are limited in their computational hardware due to payload restrictions. Thus, a map for these robots must be scalable in terms of memory and in terms of computation costs. In many cases, the creation of a global metric map is not one of the mission objectives. The robot only needs to be able to traverse between meaningful workspaces, the exact routes for getting there are not relevant. Hence, the robot can use an internal representation of the world that only contains the information that are essential for the robot to find its way. Such an internal map does not have to be Cartesian. By combining local metric navigation for obstacle avoidance with a global non-metric navigation, the robot would be able to autonomously move through complex, unknown environments.

Animals and humans are in general neither able to accurately sense metric information, nor to build correct Cartesian maps, but they can robustly find their ways through large-scale environments. Especially the navigational toolkit of insects performs remarkably well. Experiments with desert ants and bees show that they can memorize visual information of the environment [11], [12]. Hence, appearance based navigation techniques have been developed, where the robot memorizes images or special image properties in a topological map during a training phase and then navigates by matching the stored information with the current view [13], [14], [15], [16]. However, these approaches do not offer a proper scaling method, either.

Aiming at a scalable topological representation of the environment, Augustine, Mair, et al. [1], [17] introduced the LT-Map (Landmark-Tree Map) for homing based on bearing-

This work was not supported by any organization

The authors are with the Institute of Robotics and Mechatronics, German Aerospace Center (DLR), Oberpfaffenhofen, Germany annett.stelzer@dlr.de

only landmark observations. Landmark views are organized in a tree where slowly changing, translation invariant landmarks are located towards the top of the tree while translation variant landmarks are located in the leaves. The LT-Map can be scaled by pruning the tree and, thus, discarding the information about quickly changing landmarks. However, the drawback of the tree is that the resulting map structure highly depends on the order in which the landmarks are observed.

Inspired by the LT-Map concept, we propose a novel data structure, the so-called *Tra*nslation *I*nvariance *L*evel Map (*Trail-Map*), which tries to circumvent these effects by using a list structure instead of a tree. The Trail-Map better represents the degree of translation invariance of the landmarks and, hence, enables more effective pruning in case of memory shortage. As the experiments reveal, this comes with the benefit of significantly more efficient maps. In the presented cases the Trail-Map only requires about a third of the memory as occupied by an LT-Map for the same environment. After pruning, memory savings of more than 80% can be achieved, while also saving computation time.

This paper is organized as follows. Section II will briefly introduce the LT-Map concept and will show the shortcomings of this approach. In Section III, the Trail-Map and its implementation are described. Section IV will compare the performance of the Trail-Map and the LT-Map in terms of runtime, memory and navigation accuracy. Finally, Section V will conclude this paper and give an outlook on future work.

II. The LT-Map Data Structure

Since the Trail-Map data structure presented in this paper is closely related to the LT-Map, this section will briefly restate the idea behind the LT-Map concept. A more detailed description of the LT-Map is given by [17].

The LT-Map was inspired by works on insect navigation, in particular the snapshot model of [18]. This model stores the configuration of the surrounding landmarks in a snapshot. For navigation, the insect moves in a direction to align the currently perceived landmark configuration with the snapshot. This enables them to traverse a previously visited path without metric distance information and without having an explicit metric representation of the route.

In the LT-Map, a viewframe is defined as a unique configuration of landmark views that corresponds to a distinct location in space. A landmark view (LV)[1] is given by the landmark's descriptor and the unit vector pointing in the direction of the landmark. An external compass allows the rotational alignment of different viewframes.

The LT-Map is created during a mapping phase. A robot equipped with an omnidirectional sensor moves through the world, either autonomously or remotely controlled by an operator. It acquires new viewframes whenever the dissimilarity measure δ_t between the current landmark configuration and the previously acquired viewframe exceeds a specified threshold. The dissimilarity measure was computed as the average deviation of the landmark configurations between two viewframes weighted by a Pseudo-Huber cost function. Thus, the density of the viewframes depends on the local landmark configuration: In areas with only few close landmarks, the viewframes will be further apart than in areas with many nearby landmarks. That leads to an implicit adaption of the map resolution to the local conditions.

The main idea behind the LT-Map is to build up a tree structure instead of storing a list of viewframes containing all LVs. In that tree, LVs which do not change significantly between consecutive viewframe positions are shared between these viewframes. As a result, the LVs are sorted by their degree of translation invariance. LVs which do not change over many viewframes are on a higher level in the tree and correspond to translation invariant objects. LVs which change significantly from viewframe to viewframe are in the leaves and lower nodes of the tree. These LVs correspond to close objects, which are translation variant. Viewframes can be retrieved from the tree by following the path from a leaf to the root node and collecting the LVs of all visited nodes.

During the navigation phase, the robot tries to follow the previously recorded path by moving in the direction that matches the currently perceived viewframe with the next goal viewframe from the LT-Map. The navigation vector $\boldsymbol{h}$ is computed by the secant method of the difference vector model [19] as

$$\boldsymbol{h} = \frac{1}{N} \sum_{i=1}^{N} \left(\boldsymbol{l}'_i - \boldsymbol{l}_i \right) \ , \tag{1}$$

where $\boldsymbol{l}'_i$ are the unit landmark vectors in the current view and $\boldsymbol{l}_i$ are the unit landmark vectors of the goal view. Here, the sum of the difference vectors is normalized by the number of landmarks N in the viewframe.

The LT-Map data structure is claimed to be memory efficient because LVs that stay constant over several viewframes are only stored once[2]. When the angle towards a landmark changes more than a given threshold, a new LV is stored with the new landmark descriptor, what allows stable matching. Furthermore, the LT-Map data structure is scalable. When the robot runs out of memory, the leaves of the tree can be pruned. This means, that local information is discarded but the translation invariant, global information is preserved. Hence, the robot will not follow the original path exactly, but it will still be able to find its way. Only the probability of the robot to get lost increases with pruning.

A. Shortcomings of the LT-Map

Fig. 1 shows an example of a landmark tree created by the original LT-Map algorithm. The angle threshold for creating a new LV was set to $\delta_{\text{angle}} = 1^\circ$. When viewing the tree in Fig. 1 in more detail, we can see that some LVs are split into different nodes although their bearing angle difference is smaller than the assumed threshold of 1°. For example, the nodes 6 and 7 both contain the LV L_2 with angles of

[1] We will make use of the term *landmark view* (LV) in this paper to avoid confusion with the actual landmarks. In the original publication of the LT-Map the term *landmark* is used for the combination of a landmark's descriptor and its unit vector.

[2] The next section will show that this claim does not always hold.

21.3° and 21.9°, respectively. Furthermore, the LV L_3 is split between nodes 7 and 8, and the LV L_4 is split between nodes 7 and 9, although their angle differences are smaller than the threshold. Hence, the intuition that landmarks whose bearings change slowly (translation invariant) always appear in the upper levels of the tree and landmarks with quickly changing bearings (translation variant) are located in the lower levels, is not always correct. This behavior is caused by the structure of the tree. A LV is only shared by more than one viewframe if the viewframes also share all higher-level LVs. Since the angles of LVs L_3 and L_4 change from viewframe 3 to 4, the lower LV L_2 is not considered. That means, it is not even checked whether the bearing of L_2 has changed at all. Hence, the height of the LVs in the tree does not give reliable information on the level of translation invariance. For this reason, the pruning operation is likely to preserve information of volatile landmarks while discarding information of stable, translation invariant landmarks.

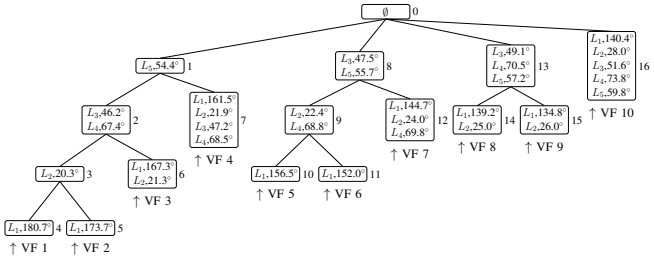


Fig. 1. Example of an LT-Map: The nodes contain the landmarks L_i with their angle when inserting them into the tree. The nodes are numbered for referencing. VF: viewframe

Hence, a tree does not seem to be the optimal data structure for the viewframe map concept, because it does not reflect the structure of an ideal viewframe map. A tree always represents a hierarchy of information and does not permit overlaps between neighboring branches. For the viewframe map, such overlaps are required. For this reason, a data structure different from a tree is better suited for representing the levels of translation invariance.

III. The Trail-Map

To come up with an alternative data structure for the viewframe map problem, let us have a look at Fig. 2, which shows a path through a simplified environment with three landmarks. When we assume an angle threshold of $\delta_{\text{angle}} = 45°$ for mapping, each landmark spans eight sectors[3] visualized by the colored lines. Within each sector, the corresponding landmark is perceived with a bearing difference of at most $45°$. A new viewframe is acquired when the angle of at least one landmark changes more than δ_{angle}. Hence, every time the robot leaves a landmark sector, it acquires a new viewframe. In this environment, a viewframe region is defined by the intersection of three specific landmark sectors. The first three viewframe regions of the robot path are highlighted in Fig. 2. When crossing a sector border, the robot also observes a new LV. The LVs of the three different landmarks are visualized by the colored path segments.

[3]Please note that for the sake of simplicity and without loss of generality we set the angle intervals to full $45°$ intervals in this example instead of letting the angle intervals start individually at the bearing the landmark is perceived first.

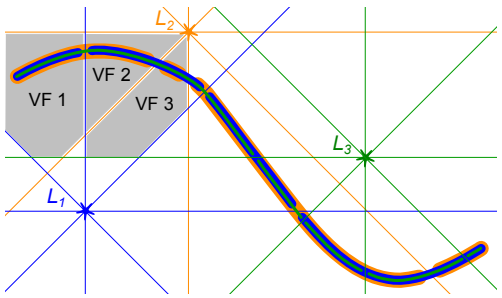


Fig. 2. Intuitive illustration of viewframe mapping. VF 1-3: highlighted first three viewframes. L_i: Landmarks with sectors

When traversing the path, the robot observes several LVs, but only one LV changes with the acquisition of each new viewframe. Hence, the robot should rather store the observed LVs instead of the single viewframes which redundantly contain many LVs. This was also the key idea behind the LT-Map. Fig. 3 illustrates the observed LVs and the corresponding viewframe numbers.

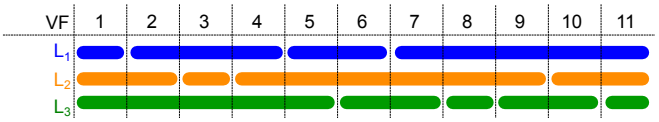


Fig. 3. From the LVs to the Trail-Map representation (step 1)

To make the viewframe map scalable in terms of memory, we want to introduce a hierarchy of how translation invariant each LV is. For this, the LVs are ordered by the number of viewframes they span. LVs that span many viewframes have a higher level of translation invariance than LVs that span only a few viewframes. LVs which span the same number of viewframes should be at the same level in the hierarchy. Fig. 4 shows the hierarchically ordered LVs. The level number gives the number of viewframes that each of the contained LVs spans. Level 4 does not exist since no LV spans exactly 4 viewframes. The resulting structure is the key idea of the translation invariance level map (Trail-Map). Now, when the robot runs out of memory, the LVs in the lower levels can be deleted without discarding stable and important long-term information.

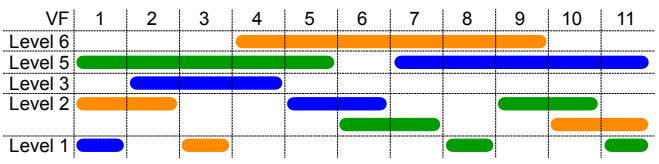


Fig. 4. From the LVs to the Trail-Map representation (step 2)

Similar to the density of the viewframes, the degree of

translation invariance and, hence, the pruning operation, also adapt to the local landmark distribution. For regions with only far landmarks, LVs in a specific level might span a much longer metric distance than LVs of the same level in regions with close landmarks. As a result, after pruning there will still be more viewframes in regions with a high landmark density than in regions with only distant landmarks.

A. Data Structure Implementation

To represent the structure of the translation invariance levels without memory overhead, we chose to use a combination of linked lists. On the top level, there is a list of translation invariance levels, which is ordered by the level number. Each level contains a linked list of LVs in the order they have been observed by the robot. A landmark view (LV) in this list consists of the landmark's descriptor, its ID, its bearing as unit vector, and the number of the viewframe when it was added to the list. The viewframe number is required for the extraction of single viewframes from the data structure.

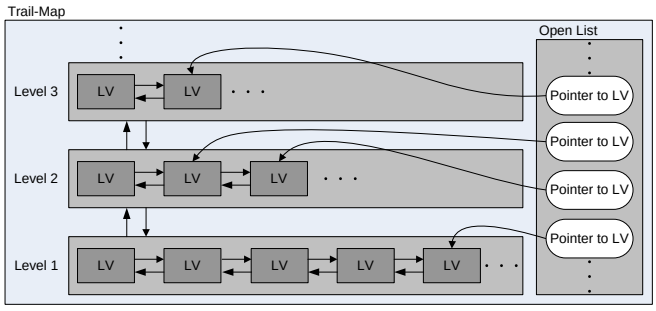


Fig. 5. Memory representation of the Trail-Map

Algorithm 1: Appending a viewframe to the Trail-Map

Input : viewframe VF; $trailMap$; $openList$
Output: $trailMap$; $openList$

```
1  if !trailMap.GetHighestLevel.IsEmpty then
2      trailMap.addLevel;
3  foreach l in openList do
4      foreach f in VF.LandmarkViews do
5          if f.descriptor matches l.descriptor then
6              if |f.angle-l.angle| < δ_angle then
7                  trailMap[l.getLevel].remove(l);
8                  trailMap[l.getLevel+1].append(l);
9                  openList.updatePointerTo(l);
10             else
11                 openList.removePointerTo(l);
12                 trailMap[1].append(f);
13                 openList.appendPointerTo(f);
14             VF.remove(f);
15             found:=true;
16             break;
17     if !found then
18         openList.removePointerTo(l);
19 foreach f in VF.LandmarkViews do
20     trailMap[1].append(f);
21     openList.appendPointerTo(f);
```

Fig. 5 illustrates the memory representation of the Trail-Map data structure. Algorithm 1 gives pseudo-code for creating this list of landmark view lists. In the beginning, the Trail-Map contains only one level: level 1. Furthermore, there is an *open list* which always contains references to the landmarks that the robot observed in the previous step. The open list is also empty initially. When the robot observes the first viewframe, all LVs of the features in the viewframe are added to level 1 and references to all these LVs are stored in the open list (lines 19-21). When the robot observes the next viewframe, level 2 is created (lines 1-2) and all LVs of the open list are compared to the features in the current viewframe (lines 3-5). If a landmark in the open list is also present in the current viewframe, and if its angle is within the angle threshold, then the LV is removed from its current level and appended to the end of the next higher level (lines 7-8). The reference to that LV in the open list is updated (line 9). If the angle of the landmark has changed more than the specified threshold, then the new LV is added to level 1, the reference to the old LV in the open list is deleted and the reference to the new LV is added at the end of the open list (lines 11-13). If a landmark in the open list cannot be found in the current viewframe, then its reference is deleted from the open list (line 18). All landmarks of the current viewframe that were observed for the first time are added to level 1 and their references are appended to the open list (lines 19-21). By repeating this procedure for each newly acquired viewframe, we achieve the following properties for the Trail-Map:

1) The level each LV is stored in corresponds to the number of viewframes for which the LV did not change its angle by more than the specified threshold δ_{angle}.
2) The references in the open list are always ordered by the number of the corresponding landmark's level.
3) The LVs in the single levels are ordered by the viewframe number they were first observed at.

Compared to the LT-Map, the Trail Map has some advantages:

- *Intuitive:* The resulting structure of the Trail-Map corresponds to the expected hierarchy of the LVs.
- *Non-redundant:* A landmark view is stored only once as long as the landmark's bearing does not change significantly.
- *Efficient:* The Trail-Map operations, especially the map pruning, are faster (ref. Section IV-B).

As a disadvantage, the retrieval of the viewframes from the Trail-Map has become more difficult. While for the LT-Map, only the paths from the leaves to the root node have to be traversed, full lists must be searched when looking for a specific viewframe number in the Trail-Map. However, since the lists are sorted, this search can be implemented efficiently. Furthermore, for general navigation it is not required to extract specific viewframes, but in most cases the viewframes are retrieved in the same or in reverse order of their visit. This stepwise retrieval can be solved with minimal costs due to the sorted structure of the lists that emerges from the construction algorithm. That means, when having retrieved one viewframe, to get the next or the previous

viewframe one only has to increment or decrement an iterator on each level list and check whether the sum of the level number and the insertion frame number of that LV fits to the number of the viewframe to be retrieved. This procedure is repeated until the newly acquired viewframe contains at least one different LV than the previous one. Furthermore, viewframes with too few LVs are discarded to allow robust navigation.

When landmarks are occluded or cannot be detected in some images, translation invariant landmarks would be inserted in lower levels due to Algorithm 1. To prevent this, landmarks which are not visible in some images and reappear at the same bearing angle should be inserted such as they were observed all the time. To do this, LVs which disappear in the next viewframe, are held in the open list and marked as *waiting*. They are only deleted from the open list, when they have not been observed for more than a specified number of viewframes or when they are observed at a significantly different angle. In case they are detected again at an angle within the angle threshold, they are moved up to the level where the landmark would have been if it had been visible all the time.

The same landmark configuration that resulted in the example LT-Map in Fig. 1 gives the Trail-Map structure depicted in Fig. 6. This structure contains 31 LVs as compared to 33 in the LT-Map. Furthermore, a new LV is only inserted when the angle changes significantly, irrespective of how the other LVs in the same viewframe behave. Thus, inconsistencies as pointed out for the LT-Map do not occur. When pruning the map in Fig. 6 by one level, the viewframes 1 and 2, and 5 and 6, respectively, will be regarded as one viewframe by the viewframe retrieval algorithm, because they contain the same LVs.

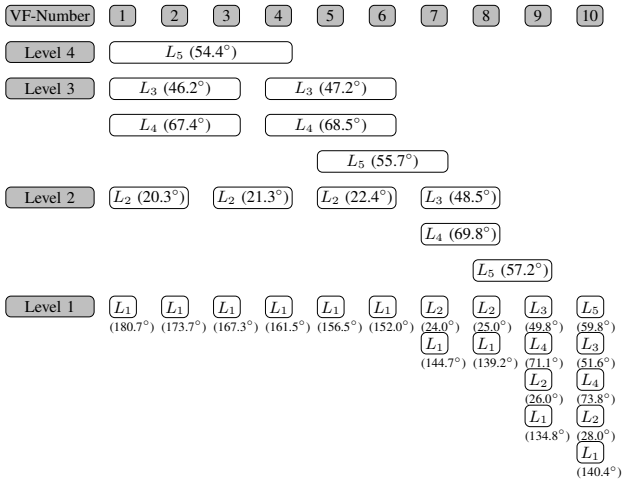


Fig. 6. Resulting Trail-Map

IV. Performance Comparison of the Trail-Map and the LT-Map

A. Memory and Navigation Performance

To compare the performances of the Trail-Map and the LT-Map, we created a simulation environment consisting of 100 uniquely identifiable landmarks spread randomly within an area of 200×200 units. A path starting at $(0, 0)$ with a length of about 130 units was defined by ten different waypoints (see Fig. 8). The simulated robot with an omnidirectional landmark sensor was lead along this path and recorded viewframes. A new viewframe was acquired as soon as one landmark changed its angle by more than δ_{angle} compared to the last recorded viewframe. We chose this measure for viewframe dissimilarity to prevent that a high number of distant landmarks suppresses the bearing changes of local landmarks, what might happen by calculating an average angle deviation of the landmarks as proposed in [17]. The smaller the value of δ_{angle}, the higher the resolution and, hence, the accuracy and the memory requirements of the resulting map. The value of δ_{angle} should be higher than the resolution of the bearing sensor and should be chosen according to the required path accuracy. Since usually no high accuracy is required for the traversal between different workspaces, higher thresholds should be preferred for the benefit of less memory. In these simulations, we chose $\delta_{\text{angle}} = 5°$.

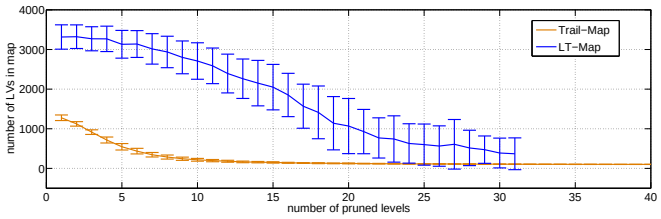
We used the secant method as previously stated in Eq. 1 for calculating the navigation vector direction. However, the length of the navigation vector was adapted to the local landmark configurations. For $\Delta_{\text{max}} < \delta_{\text{angle}}$ the navigation vector length was increased, for $\Delta_{\text{max}} > 2\delta_{\text{angle}}$ it was decreased, where Δ_{max} is the maximum change of a landmark angle between two navigation steps.

Thus, in regions with nearby landmarks, the single navigation steps were smaller than in regions with landmarks that were far away. We simulated no noise, outliers, or occlusions, because the basic behavior of both approaches should be compared. Based on the recorded viewframes, the simulated robot should navigate along the same path using the LT-Map and the Trail-Map at different pruning levels. To ensure that the goal point can always be reached with a good accuracy, the last viewframe acquired at the goal point was never pruned. The simulation was run 100 times with different isotropic landmark configurations for each pruning level and the landmark configuration was randomly changed after each run. The path error is computed as the area between the learning and the navigation trajectory.

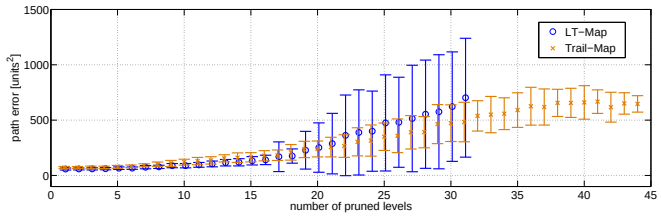
Fig. 7 visualizes the statistics of the simulation results. One of the most noticeable things is that the full LT-Map contains 3315 LVs on average while the Trail-Map requires 1278 LVs, which is only 38.5% (ref. Fig. 7(a)). That shows, that the Trail-Map data structure is much more memory efficient because LVs are not stored redundantly.

When pruning the Trail-Map, the number of LVs reduces quickly. In contrast, when pruning the LT-Map, in the beginning only a few deep branches are affected. This shows, that the LT-Map is not as well-balanced as the Trail-Map.

The path error resulting from the full LT-Map is somewhat smaller than the path error of the full Trail-Map, as shown in Fig. 7(b). The reason for that is the already mentioned fact that the LT-Map often includes LVs again with their current angle, because higher-level LVs have changed, although the



(a) number of LVs



(b) path error

Fig. 7. Statistical comparison of LT-Map and Trail-Map depending on the number of pruned levels (means and standard deviations)

lower LVs have not exceeded the threshold δ_{angle} yet. Hence, in the LT-Map the landmark angle resolution is on average higher than $\delta_{\text{angle}} = 5°$, what results in a more accurate navigation path. What should be noted for interpreting the

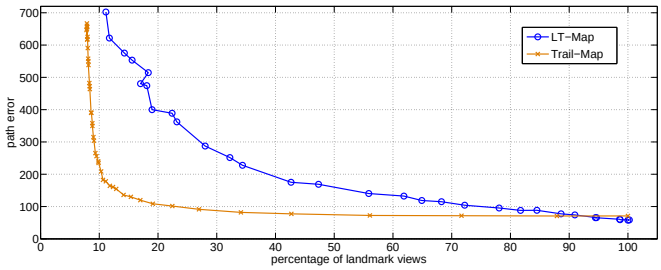


Fig. 8. Comparison of path errors of LT-Map and Trail-Map for different pruning ratios

data in Fig. 7, is the fact that for the 100 simulated landmark configurations, not all resulting LT-Maps had the same height. That means, after pruning 20 levels or more, only a fraction of the created LT-Maps could be considered in the calculation of the mean, because many LT-Maps had no LVs left. The same applied to the resulting Trail-Maps, because not all Trail-Maps had more than 30 levels. Furthermore, the standard deviations of the number of LVs in the LT-Map grow very big when pruning many levels, because the LT-Map contains the most LVs towards the root of the tree. The standard deviations of the number of LVs in the Trail-Map show opposite behavior because in the Trail-Map most LVs are located in the lower levels. We are aware of the fact, that the distributions of the path error and the number of LVs are not always Gaussian, but we use the standard deviations in the error bar plots to give an impression of how much the different values vary.

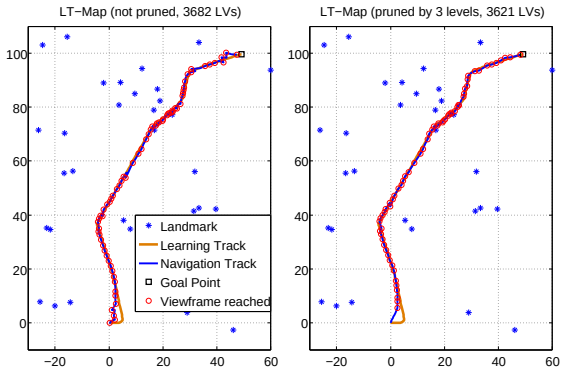
The most remarkable plot is Fig. 8, which illustrates the average behavior of the path error depending on the percentage of remaining LVs. The navigation performance of the LT-Map degrades quickly. The path error of the LT-Map exceeds the minimum path error of the Trail-Map after pruning about 10% of the LVs. In contrast, the Trail-Map can be pruned to about 50% of the original number of LVs without significant loss of path accuracy. This advance in performance of the Trail-Map is remarkable, keeping in mind that the 100% mark corresponds to 3315 LVs for the LT-Map, but only to 1278 LVs for the Trail-Map. Comparing the number of required LVs to achieve a path error of about $\epsilon = 77$, the LT-Map requires 2938 LVs, while the Trail-Map only needs 546 LVs on average. That means a memory saving of more than 80%. Considering path errors of about $\epsilon = 160$, the Trail-Map saves about 90% of LVs compared to the LT-Map.

Fig. 9 shows the navigated paths for different pruning levels for one example environment. The resulting Trail-Map contains 38 levels in this case, the lowest being 1 and the highest being 50. The tree of the LT-Map has a maximal height of 26. Fig. 9(a) and Fig. 10(a) show the navigated paths using the full LT-Map and the full Trail-Map, respectively. The accuracy of the followed trajectories is approximately similar in this example. However, when the LT-Map is pruned to the number of landmarks of the full Trail-Map, the navigated path already degrades significantly (see Fig. 9(e)). As Fig. 9(f) shows, with 600 LVs the resulting path of the LT-Map has only very few viewframe locations left, hence, the accuracy decreases noticeably. In contrast, when the Trail-Map is pruned to the same proportion of its original size (see Fig. 10(d)), viewframes are still spread over the whole path. Even with as few as 153 LVs (which is 11% of the original number), the path is still followed well.

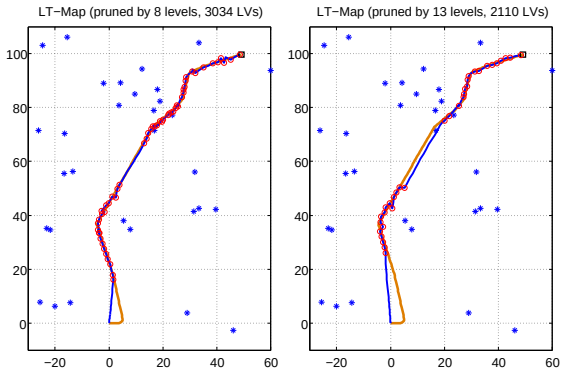
When comparing the navigation trajectories of the LT-Map and the Trail-Map using only the last viewframe (Fig. 9(h) and Fig. 10(h)), it becomes obvious that the LT-Map navigation track is a nearly direct path to the goal point, while the Trail-Map track still goes through intermediate viewframes. The reason for this is, that in the Trail-Map the last viewframe still contains the information about the levels of each LV. LVs of the higher levels produce intermediate viewframes which are reached first. Later, LVs in the lower levels create refined viewframes that finally lead to the goal position. This kind of information is lost when pruning the LT-Map tree.

B. Runtime Performance

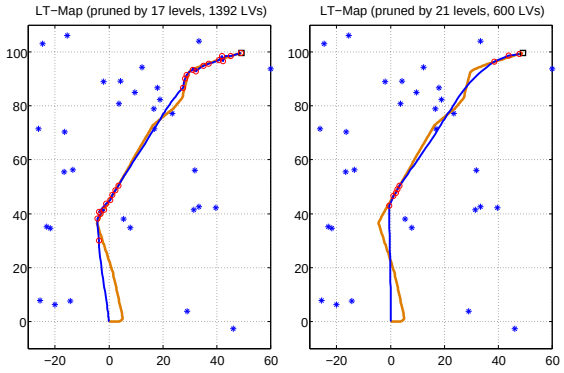
To show the computational efficiency of the Trail-Map, we compared the runtimes of our Trail-Map implementation and the original implementation of the LT-Map. Both maps were implemented in C++ and compiled with the same compiler and flags. Runtimes are measured on a 2.67 GHz CPU. We used the simulation environment and path described in the previous section and created environments with 100 up to 5000 randomly distributed landmarks spread within the range of 200×200 units. The recorded viewframes were imported into the C++ programs and the runtimes for creating full maps from the recorded viewframes, for map pruning and for viewframe retrieval were measured.



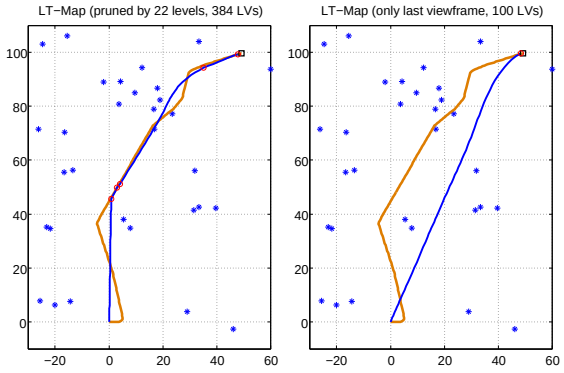
(a) 100% LVs, $\epsilon = 54.7$ (b) 98% LVs, $\epsilon = 52.9$



(c) 82% LVs, $\epsilon = 72.7$ (d) 57% LVs, $\epsilon = 156.6$

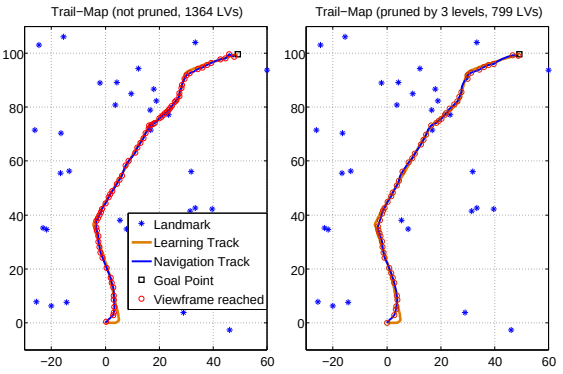


(e) 38% LVs, $\epsilon = 174.8$ (f) 16% LVs, $\epsilon = 175.2$

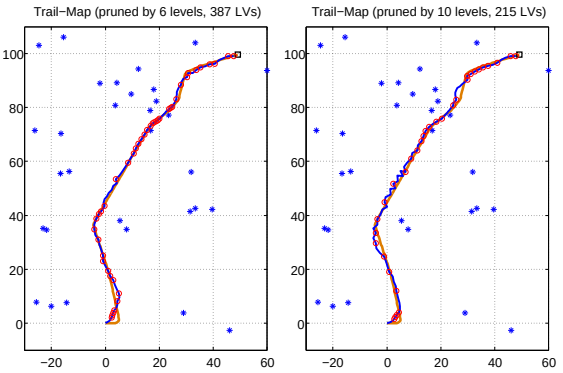


(g) 10% LVs, $\epsilon = 195.9$ (h) 3% LVs, $\epsilon = 1204.2$

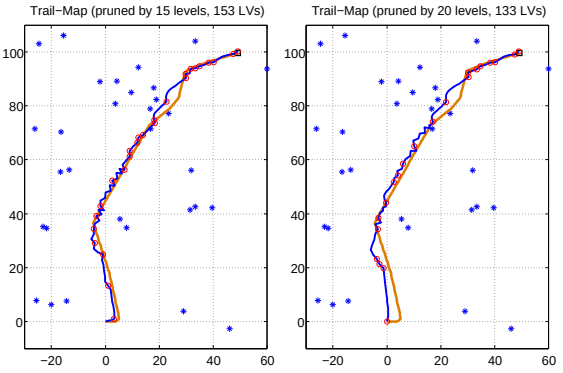
Fig. 9. LT-Map pruning behavior (ϵ: path error)



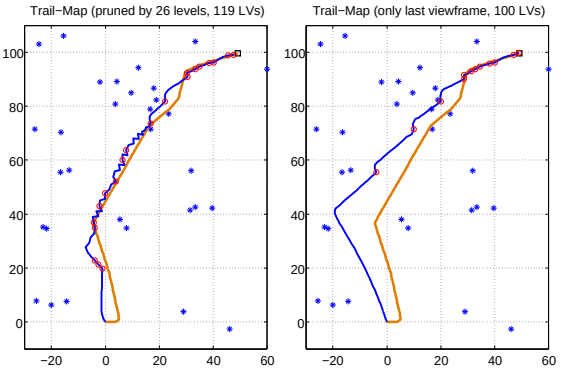
(a) 100% LVs, $\epsilon = 55.5$ (b) 59% LVs, $\epsilon = 56.0$



(c) 28% LVs, $\epsilon = 61.9$ (d) 16% LVs, $\epsilon = 99.5$



(e) 11% LVs, $\epsilon = 142.7$ (f) 10% LVs, $\epsilon = 236.9$



(g) 9% LVs, $\epsilon = 262.5$ (h) 7% LVs, $\epsilon = 699.9$

Fig. 10. Trail-Map pruning behavior (ϵ: path error)

According to Algorithm 1, a viewframe is added to the Trail-Map in O(N^2) with N being the number of LVs per viewframe. The worst case complexity of the LT-Map creation is also O(N^2), but since the landmark comparison with the previous viewframe can stop as soon as a LV changes significantly, the best case complexity of creating the LT-Map is O(N). Since N stays constant during the mapping phase, the runtime for map creation does not grow with the map. As Fig. 11 shows, the Trail-Map can be created faster than the LT-Map when the viewframes do not contain more than about 1000 LVs, what is sufficient for most applications.

Additionally, the Trail-Map can be pruned magnitudes faster than the LT-Map. Especially when pruning only one level, the LT-Map needs to traverse the full tree to find the leaves which have to be pruned, while only one level list has to be deleted from the Trail-Map. For 5000 LVs per viewframe, pruning takes up to 350 ms for the LT-Map, but less than 1 ms for the Trail-Map. For 1000 LVs, the LT-Map can be pruned in 30 ms, the Trail-Map in 70 ns.

Furthermore, the retrieval of the viewframes from the created Trail-Maps is faster than from the LT-Maps because, although the number of LVs per viewframe is the same, the LT-Maps contain more LVs than the Trail-Maps, due to redundancies.

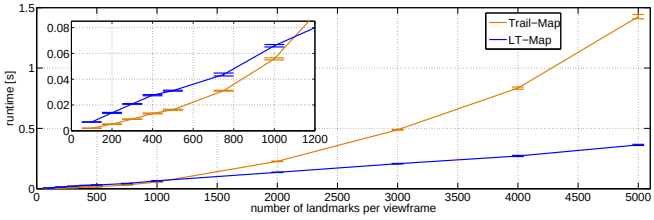


Fig. 11. Map creation time comparison of LT-Map and Trail-Map

V. Conclusion

In this paper, we presented the Trail-Map, a novel data structure for range-free homing. The Trail-Map is closely related to the LT-Map, in particular it borrows the idea to order landmarks hierarchically by their level of translation invariance. This allows discarding the least important landmark views in case of memory shortage. However, in comparison to the LT-Map, the structure of the Trail-Map is better suited for viewframe-based navigation and hence consumes much less memory. Additionally, it can be pruned stronger without significant loss of accuracy. Precisely, more than 80% of memory can be saved in our experiments compared to the LT-Map, while also saving computation time.

On top of a local navigation scheme for obstacle avoidance and local planning, the Trail-Map enables global navigation in unknown environments where the robot has to traverse previously visited paths again, for example when returning to its starting position. The proposed navigation scheme adapts to local landmark configurations by creating more viewframes in areas with close landmarks and less viewframes in areas with far landmarks. It does not aim at a very high navigation accuracy but at robustly following a path with as little memory and computational costs as possible. The Trail-Map can be created and pruned efficiently, what makes it well suited for mobile robots with limited hardware resources, such as planetary rovers or MAVs.

Next, we are going to evaluate the performance of the Trail-Map with natural landmarks in real-world experiments.

References

[1] M. Augustine, E. Mair, A. Stelzer, F. Ortmeier, D. Burschka, and M. Suppa, "Landmark-Tree Map: A biologically inspired topological map for long-distance robot navigation," in *IEEE International Conference on Robotics and Biomimetics (ROBIO) 2012, Dec. 11-14, 2012, Guangzhou, China*, 2012.

[2] D. Pai and L. Reissell, "Multiresolution rough terrain motion planning," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 1, pp. 19–33, 1998.

[3] C. Plagemann, S. Mischke, S. Prentice, K. Kersting, N. Roy, and W. Burgard, "A bayesian regression approach to terrain mapping and an application to legged robot locomotion," *Journal of Field Robotics*, vol. 26, no. 10, pp. 789–811, 2009.

[4] J. Nieto, J. Guivant, and E. Nebot, "DenseSLAM: simultaneous localization and dense mapping," *International Journal of Robotics Research*, vol. 25, no. 8, pp. 711–744, 2006.

[5] K. Konolige, M. Agrawal, R. Bolles, C. Cowan, M. Fischler, and B. Gerkey, "Outdoor mapping and navigation using stereo vision," in *Proc. of the International Symposium on Experimental Robotics (ISER)*, 2007.

[6] S. Se, D. Lowe, and J. Little, "Mobile robot localization and mapping with uncertainty using scale-invariant visual landmarks," *International Journal of Robotics Research*, vol. 21, no. 8, pp. 735–758, 2002.

[7] 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: AAAI, 2002.

[8] S. Thrun, Y. Liu, D. Koller, A. Ng, Z. Ghahramani, and H. Durrant-Whyte, "Simultaneous localization and mapping with sparse extended information filters," *The International Journal of Robotics Research*, vol. 23, no. 7-8, p. 693, 2004.

[9] S. Thrun, "Learning metric-topological maps for indoor mobile robot navigation," *Artificial Intelligence*, vol. 99, no. 1, pp. 21–71, 1998, hybrid Metric Topological.

[10] J. Blanco, J. Fernandez-Madrigal, and J. Gonzalez, "Toward a unified bayesian approach to hybrid metric-topological SLAM," *IEEE Transactions on Robotics*, vol. 24, no. 2, pp. 259–270, 2008.

[11] M. Collett, "Spatial memories in insects," *Current Biology*, vol. 19, no. 24, pp. R1103–R1108, 2009.

[12] R. Wehner, "Desert ant navigation: How miniature brains solve complex tasks," *Journal of Comparative Physiology A*, vol. 189, no. 8, pp. 579–588, 2003.

[13] Y. Matsumoto, M. Inaba, and H. Inoue, "Visual navigation using view-sequenced route representation," in *Proc. of the IEEE International Conference on Robotics and Automation*, vol. 1, 1996, pp. 83–88.

[14] S. D. Jones, C. Andresen, and J. Crowley, "Appearance based process for visual navigation," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 2, 1997, pp. 551–557.

[15] N. Winters and J. Santos-Victor, "Omni-directional visual navigation," in *Proc. of the 7th International Symposium on Intelligent Robotic Systems (SIRS 99)*, 1999, pp. 109–118.

[16] L. Kosecka, J.and Zhou, P. Barber, and Z. Duric, "Qualitative image based localization in indoors environments," in *Proc. of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2. IEEE, 2003, pp. II–3.

[17] E. Mair, M. Augustine, B. Jäger, A. Stelzer, C. Brand, D. Burschka, and M. Suppa, "A biologically inspired navigation concept based on the landmark-tree map for efficient long-distance robot navigation," *Advanced Robotics*, vol. 28, no. 5, pp. 289–302, 2014.

[18] B. Cartwright and T. Collett, "Landmark maps for honeybees," *Biological Cybernetics*, vol. 57, no. 1-2, pp. 85–93, 1987.

[19] D. Lambrinos, T. Labhart, and R. Pfeifer, "A mobile robot employing insect strategies for navigation," *Robotics and Autonomous Systems*, vol. 30, no. 1-2, pp. 39–64, 2000.