Published at the Workshop on State Estimation and Terrain Perception for All Terrain Mobile Robots held in conjunction with the International Conference on Intelligent Robots and Systems (IROS), 2016.

Properties of timebased local OctoMaps

Peter Weissig and Peter Protzel

***Abstract*— Autonomous navigation of our rough-terrain rovers implies the need of a good representation of their near surrounding. In order to archive this we fuse several of their sensors into one representation called OctoMap. But moving obstacles can produce artefacts, leading to untraversable regions. Furthermore the map itself is increasing in size while discovering new places. Even though we are only interested in the near surrounding of the rovers.**

Our approach to these problems is the usage of timestamps within the map. If a certain region was not updated within a given interval, it will be set to free space or deleted from the map. This first option is an existing solution and the second option reflects our new alternative.

The proposed approach will be provided as open source[1].

I. MOTIVATION

Within our group we focus on autonomous mapping and navigation, especially on rough terrain and without GNSS. In the following figure 1 are two of our rovers equipped with a robust skid-steering chassis and different sensors.

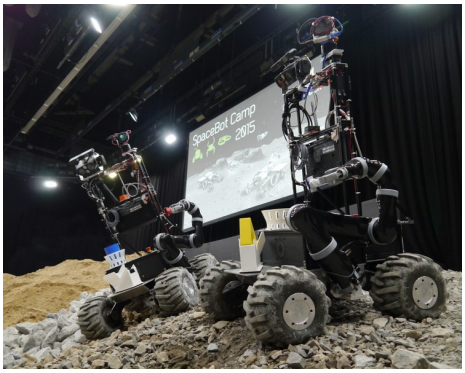

Fig. 1. Two of our rovers during the spacebot cup [1].

On one hand we use the multisense S7[2] stereo camera and the Xtion[3] RGBD-sensor combination as visual depth sensor. This gives us dense pointclouds with a high update rate, however those are only covering a small part of the surrounding. On the other hand we use a rotating laser-scanner [2], leading to accurate, omni-directional and long-distance, measurements. Nevertheless a full scan takes several seconds.

In order to use the advantages of both sensor types we fuse them using the OctoMap [3], a discrete, efficient and probabilistic representation. For each inserted measurement the related voxels are updated as *occupied*. In additon all voxels along the ray from the sensor to the detected obstacles must be free. Therefore they will be updated as *free*.

Using this plain OctoMap a couple of problems emerge:

- Voxels are never deleted. While driving around, the map continuously increases in size. Although we only rely on the continuously-updated close surrounding of the rover.
- Voxels are only cleared if they are seen as ”free”. As a result a resting rover may experience problems. For example a moving object like a second rover may be seen instantaneous. In contrast it is never cleared from the map, if there is no obstacle within the range of the sensors.
- Also the area directly under each rover is unseen by any sensor. Thus a bad motion estimation may lead to a rover taking off or diving into the ground. Which in turn disturbs the path planner.

To address all of these problems we compare two types of timebased local OctoMaps. Both handle voxels that were not updated for a certain amount of time. One type was implemented by the authors of the OctoMap as *class OcTreeNodeStamped*[4]. The other is our alternative implementation, having different properties.

II. IMPLEMANTATIONS

The basic OctoMap has three important properties: it is discrete, efficient and probabilistic. Discrete means that the whole volume covered by the map is represented by cubic cells, called voxels. The voxels are the smallest unit for which distinct properties can be set. The map is efficient since the voxels are saved as a tree structure and not as a fixed grid. Finally the map is probabilistic because it is storing probabilities of occupancy instead of binary states, like free and occupied. These probabilities are usually updated by new measurement using the rule of bayes.

Both timebased implementations extent the basic OctoMap by storing an additional timestamp for each voxel. For new measurements related voxels will update their occupancy and their timestamp. The map can therefore degrade outdated voxels as needed.

The first implementation was created by authors of [3]. Their solution is using timestamps based on the local time of the OctoMap. This may lead to unwanted effects while playing back from recorded data-files at none-realtime. Also the insertion of one measurement will create many slightly different timestamps. Furthermore the degradation of voxels

The authors are with Technische Universität Chemnitz, Germany firstname.lastname@etit.tu-chemnitz.de

[1] http://tu-chemnitz.de/etit/proaut/octo

[2] http://carnegierobotics.com/multisense-s7/

[3] http://www.asus.com/de/3D-Sensor/Xtion_PRO/

[4] http://github.com/OctoMap/octomap/blob/master/octomap/include/octomap/OcTreeStamped.h

does only change their occupancy from *occupied* to *free*. This wrongly indicates free space instead of forgotten so called *unkown* space. In the end those voxels will remain and consume memory.

The second implementation was created by us. Our timestamp is based on the incoming sensor messages and not on the system time. Since these messages are not always in order the timestamp will updated as necessary. For us degradation of voxels does not mean changing their occupancy to free, but deleting those. This will reduce memory usage. Moreover it implies the information of not knowing anything about the voxels instead of suggesting free space. Finally the degradation is done based on the last received sensor message.

III. EXPERIMENTS

We evaluated both implementations within two scenarios. The first one is a multi-robot scenario. One rover is standing still, while a second rover is slowly crossing its close surrounding. Since our visual depth sensors are tilted downward only the lower part of the driving rover is updated almost on time. In contrast the upper part is only sensed by the slowly turning laser, creating several artifacts. As seen in figure 2 both timebased implementations remove artefacts as expected. Only those too new to be degraded are within the map for a certain amount of time.

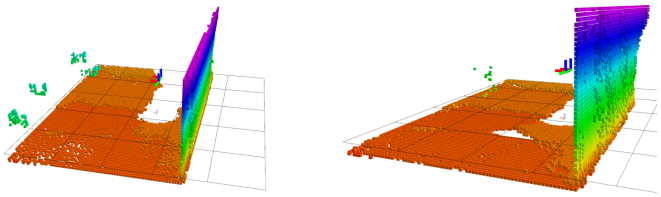

Fig. 2. Artefacts created by a crossing rover as seen as green voxels on the left side of each image. Left: The artifacts stay within the basic OctoMap. Right: Both timebased implementations removed almost all artefacts. Since their output is nearly identical, only the result from our implementation is shown.

The second scenario is an outdoor exploration. One rover is driving along a rectangular path of approximately $90\,\mathrm{m} \times 40\,\mathrm{m}$ while mapping its surrounding. The final result of both implementations regarding occupied voxels are alike and therefore not shown in figure 3. However a distinctive difference is seen when comparing free voxels. Here our implementation is clearly more memory saving.

IV. DISCUSSION

An exploring rover may create a huge map of its known world. Nevertheless we are only interested in the close surrounding, e.g. to create a local traversability map for the pathplanner. A simple solution could be the degradation of remote voxels. This will work perfectly and efficiently for the map size, however it will not handle artefacts as shown in figure 2. Therefore a timebased approach is recommanded, as it will degrade artefacts and remote voxels. On the contrary the new attribute will take some additional memory and processing time and it will reduce the likeliness of prunning.

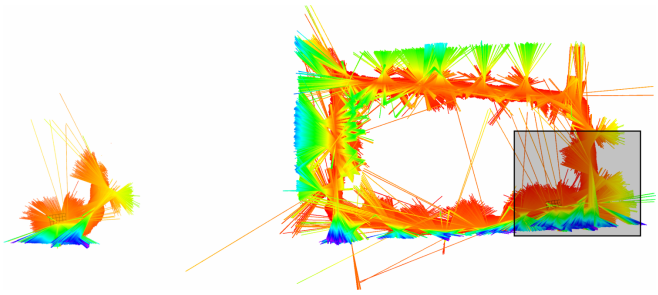

Fig. 3. Top view of the free space of the explored map. Left: Our implementation - in total about 202 Mbyte. Right: Implementation from the authors of [3] - in total about 867 Mbyte. The black rectangle shows the approximate position and size of the left image.

The timebased approach itself can also be interpreted as a temporal storage forgetting old data, which is likely to be outdated. That is why we delete those voxels. This saves memory and it implies that there is no information about the voxel anymore - it is *unknown*. Yet the deletion is timewise more costly than a simple update of the probability of occupancy. In case the rover stays within a limited region, as in scenario one, there might be many unneccessary deletions and creations.

The data within our map is mainly used to feed the local path planner of the rover. Therefore only occupied voxels are exported and processed. This means that the implementation of the authors of [3] should be sufficient. Especially as it is faster than our implementation. However their map will grow in size if the rover is exploring. Also the difference between *unknown* and *free* space is important. If a new measurement suggests an obstacle in a degraded voxel their approach may need additional measurements and hence more time to convert the voxel from *free* to *occupied*.

Regarding the kind of timestamp, our version is in advantage. It is capable of handling datastreams not synchronized to real time, e.g. if they are played back from a recorded file at different speeds. Finally all updates belonging to one measurement share the exact same timestamp and therefore prunning is more likely to happen.

V. CONCLUSION

Degradation of outdated voxels is an important extension of the OctoMap. The main difference between the implementation from the authors of [3] and our alternative is what happens to these voxels. As they set those to free space, they are faster. On the contrary we save memory by deletion.

REFERENCES

[1] S. Lange, D. Wunschel, S. Schubert, T. Pfeifer, P. Weissig, A. Uhlig, M. Truschzinski, and P. Protzel, "Two autonomous robots for the DLR SpaceBot cup lessons learned from 60 minutes on the moon." in *Int. Symp. on Robotics*, Munich, Germany, 2016.

[2] S. Schubert, P. Neubert, and P. Protzel, "How to build and customize a high-resolution 3d laserscanner using off-the-shelf components," in *Proc. of Towards Autonomous Robotic Systems*, Sheffield, England, 2016.

[3] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, 2013, software available at http://octomap.github.com. [Online]. Available: http://octomap.github.com