Towards 6D MCL for LiDARs in 3D TSDF Maps on Embedded Systems with GPUs

Marc Eisoldt[1], Alexander Mock[2], Mario Porrmann[1] and Thomas Wiemann[3]

***Abstract*— Monte Carlo Localization is a widely used approach in the field of mobile robotics. While this problem has been well studied in the 2D case, global localization in 3D maps with six degrees of freedom has so far been too computationally demanding. Hence, no mobile robot system has yet been presented in literature that is able to solve it in real-time. The computationally most intensive step is the evaluation of the sensor model, but it also offers high parallelization potential. This work investigates the massive parallelization of the evaluation of particles in truncated signed distance fields for three-dimensional laser scanners on embedded GPUs. The implementation on the GPU is 30 times as fast and more than 50 times more energy efficient compared to a CPU implementation.**

I. Introduction

Localization in maps is one of the fundamental problems in mobile robotics. Many applications like navigation or exploration rely on a known pose of a robot in a given map. Multimodal probabilistic methods are the state of the art for self-localization in GPS-denied, marker-less regions. The exact calculation of such models is too expensive, so particle filters are used to approximate the problem. Monte Carlo Localization (MCL) uses particles to sub-sample the infinite state space. Global localization with 3D poses in 2D maps can be efficiently solved with CPUs, as the number of required particles is small. In realistic applications using smart re-sampling, only a few thousand particles are required in 2D.

3D maps can be generated with LiDAR sensors in short time with high accuracy [1]. LiDAR sensors are small and lightweight, allowing them to be used on drones [2]. To localize such vehicles with MCL similar to the 3D case, would allow to develop autonomous flying systems for GPS-denied environments without relying on artificial beacons, allowing new applications for instance in autonomous maintenance and logistics. With this work, we to provide a MCL implementation for 3D maps and 6D poses. The challenge is, that for 6D poses in 3D maps, the number of required samples increases by several orders of magnitude. Currently, this problem can not be solved in real-time on standard CPUs. Particle filters offer great potential to benefit from hardware architectures that allow massively parallel computations. Accordingly, we aim to exploit such parallelism on embedded GPUs to achieve real-time localization.

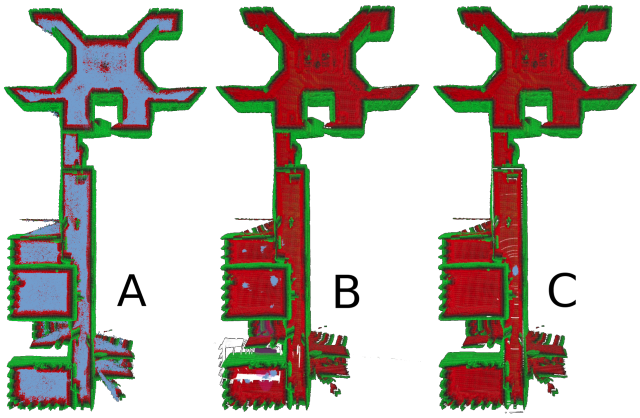

Fig. 1: Our 6D MCL implementation successfully localizes a robot in a large-scale 3D TSDF map, colored in red and green, without initial guess. The multimodal particle distribution is colored in blue and is captured during initialization (A), after several updates (B) and near conversion to the real pose (C).

Localization requires 3D map representations, that support quick simulation of hypothetical LiDAR data. For that, Truncated Signed Distance Fields (TSDF) are apt candidates. They allow to compute the required sensor update efficiently, i.e., measuring the scan-to-map divergence for a certain LiDAR measurement at a certain pose hypothesis, as the closest distances to the surface can be directly read from the TSDF.

In this paper, we present an approach that leverages those beneficial properties of TSDF maps and GPUs to achieve a degree of acceleration that allows to use MCL in 6D on embedded hardware. We provide our implementation in the open-source package `tsdf_localization`[1] that is fully integrated with ROS. In our experiments, we evaluate our software with robots equipped with a LiDAR such as a Velodyne VLP-16 or an Ouster OS0-64, and TSDF maps of various environments. Our GPU architecture is able to speed up the computation significantly compared to the CPU baseline, while consuming considerably less power. Moreover, we show that our software is not only suitable for pose tracking but also for global localization of a mobile robot as visualized in Fig. 1.

[1] Computer Engineering Group, Osnabrück University, Osnabrück, Germany `firstname.lastname@uni-osnabrueck.de`

[2] Knowledge-based Systems Group, Osnabrück University, Osnabrück, Germany `amock@uni-osnabrueck.de`

[3] Robotics in Computer Science, Fulda University of Applied Sciences, Fulda, Germany and DFKI Niedersachsen, Plan-based Robot Control, Osnabrück, Germany `thomas.wiemann@informatik.hs-fulda.de`

The DFKI Niedersachsen Lab (DFKI NI) is sponsored by the Ministry of Science and Culture of Lower Saxony and the VolkswagenStiftung

[1] Available under BSD 3-clause license: `https://github.com/uos/tsdf_localization`

II. Related Work

MCL was first introduced by Fox et al. [3]. It is an approximation of Markov localization in a 2D grid map and approximates the set of considered states of the robot with a particle filter. This significantly increases the performance of the algorithm and allows a global 2D localization with up to 5,000 particles on a CPU. Using the Kullback–Leibler divergence [4], the number of particles can be reduced by adapting the number of particles dynamically over time (KLD sampling). Besides global localization, particle filters are also used in Simultaneous Localization and Mapping (SLAM) algorithms to track the position of the robot and map the environment in 2D using various pose hypotheses [5], [6]. Because the initial pose of the robot in the map is known, e.g., Grisetti et al. [6] require only 60 particles to track a robot's position.

These approaches concentrate on the localization of robots in 2D maps of planar environments. For many applications like UAVs in GPS-denied places, localization with six degrees of freedom (DoF) is desirable. Perez-Grau et al., for instance, extended MCL to track the position of a flying robot [7]. They used landmarks for pose tracking, but did not consider the global localization problem. Even in this simplified scenario, they already required 300 particles. The global localization problem in 3D is addressed in works by Oishi et al. and Hornung et al. [8], [9]. Their method is not real-time capable and requires between 20,000 and 72,000 particles to localize a robot in a 3D map, which is ten times more compared to the 2D case. Kümmerle et al. also presented an approach for MCL in 3D to localize a robot in an outdoor environment [10]. They stated that in their scenario at least 100,000 particles are required to achieve basic localization. All these papers demonstrate the high computational demands of the problem and state that, in comparison with the 2D case, significantly more particles are required.

Due to the parallel nature of the particle filter, in recent years parallel hardware architectures have been used to speed up computations. Existing methods using such hardware only consider local tracking and can be divided into GPU-based and FPGA-based approaches. Examples for GPU-accelerated algorithms are [11]–[14]. In the experiments presented in these works, a fixed number of up to 1,000 particles has been considered, far less than required for realistic applications. Related methods using FPGAs are presented in [15]–[18]. They consider the general algorithm of the particle filter to solve the bearing-only tracking problem, which is not as complex as 6D MCL. Other existing works like [19], [20] use particle filters to track objects in camera images. Such tracking problems are less complex than global localization, hence tracking is possible with only 300 particles.

Based on this review of existing literature, it can be concluded that global MCL in 6D is still unsolved for realistic applications. We estimate, that at least 100,000 particles have to be considered when solving the problem for a realistic indoor environment. Since mobile robots usually have a limited power budget, solutions have to be found that allow to simulate such high numbers with minimal energy consumption.

In the case of particle filters applied to the problem of robot localization, GPUs have already achieved a high acceleration of the algorithm and are highly promising to get a significant step closer to the global localization in 6 DoF, where FPGA-based implementations have mostly dealt with less complex scenarios and are yet not as high investigated for this kind of problem as the GPU.

In this work, we provide an implementation for embedded GPU-based architectures to accelerate MCL in 3D TSDF maps to investigate the potential of developing a mobile robot system with 6 DoF. We provide details on the implementation, including mechanics to ensure efficient memory access, which proves to be important to avoid bottlenecks in the parallel computation. In the evaluation, we present a use-case, where a mobile robot is localized in a simulated office environment and also analyze the architecture using an established real-world benchmark dataset. We show, that our implementation is able to solve the global localization problem significantly faster while reducing the power consumption clearly compared to standard CPU-based hardware.

III. Algorithm

As our algorithm is similar to standard MCL [3], this section only briefly summarizes that approach, concentrating on the special properties of TSDF maps. They show several properties that are beneficial for our purposes. First, the map can be efficiently represented in a sparse 3D array. Second, although the representation itself is discrete, for each possible pose, a pseudo-continuous state can be estimated via interpolation. Secondly, for each hypothesis and a given scan, only a simple lookup is required to obtain the closest distance to the map, which immediately results in an update of the sensor. Considering these properties, the transfer from standard MCL to 6D MCL is straight forward, although computationally demanding.

To support fast evaluation of the sensor model, a TSDF map is used. Similar to Akai et al. [21], it is split up into

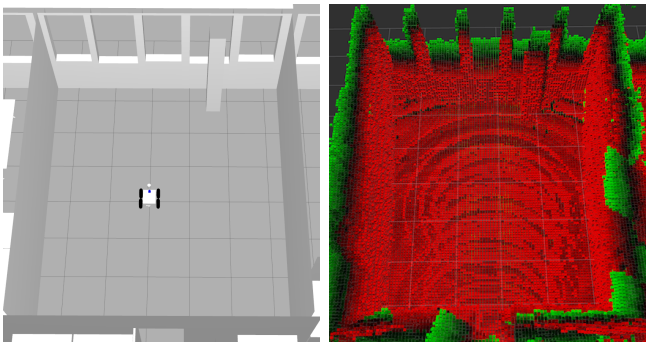

Fig. 2: Example of the simulation environment (left) with the robot model and the resulting TSDF volume (right). Red cells represent positive values and green cells represent negative values. Darker cells are closer to the surface. Cells exceeding the truncation distance are not rendered.

a fine and a coarse-grained representation to save memory while allowing fast access. The map only holds parts in the coarse grid, which contain distance values within the truncation distance. For all other grid cells, the values are clamped to the truncation value and can be neglected. Hence, the presented method scales well to larger environments. An example of such a map is shown Fig. 2.

In the first step of the algorithm, the particle set is initialized based on the given initial distribution of the robot's state. Each particle is represented as described in Eq. 1 and Eq. 2. It holds the robot state s_t in 6D (position and rotation) and a weight w_t.

$$p_t^{[n]} = (s_t^{[n]}, w_t^{[n]}) \tag{1}$$

$$s_t^{[n]} = (x_t^{[n]}, y_t^{[n]}, z_t^{[n]}, \phi_t^{[n]}, \psi_t^{[n]}, \theta_t^{[n]}) \tag{2}$$

If a pose estimation is available, the particles are distributed with predefined variance around the initial state estimation. Without an initial pose estimate, the particles are distributed uniformly in free space. After initialization, three steps of the algorithm are repeated to iteratively improve the pose estimation of the robot. The first step is the motion update that applies the motion model to each particle.

After the propagation of the robot's state, the sensor update is computed, where the likelihood for every particle is calculated using the sensor model of the robot. In this step, a virtual sensor measurement for every particle is generated and compared to the map. The weight of a particle increases with the match of the observation to the environment. In this work, the endpoint model is used to estimate the weights, where the distance of the scan points to the surface of the environment is used as measure of the match. Because of the structure of the TSDF representation, this can be calculated directly using Eq. 3. The weight for every particle depends on the considered pose x_t in the map m. $m(x_t, z_t)$ represents the signed distance of the considered scan point z_t from the perspective of the particle to the next obstacle in the map, which can be obtained directly by a look-up in the TSDF volume. All entries of the map can be calculated before the start of the scan procedure, where the inefficient calculation of the exponential function can be prevented. However, using a laser scanner the sensor update is still computation expensive, because a high number of scan points must be evaluated for every single particle.

$$p_{\text{hit}}(z_t | x_t^{[n]}, m) \approx \frac{1}{\sqrt{2 \cdot \pi \cdot \sigma^2}} \exp\left(-\frac{1}{2} \cdot \frac{m(x_t^{[n]}, z_t)^2}{\sigma^2}\right) \tag{3}$$

The pose can then be estimated by the weighting average of the particles or by using the particle with the highest weight as the current pose of the robot. The last step is to draw a new set of particles. This is done using the state distribution represented by the current particle set. Based on the calculated weights, a new set of particles is chosen from the previous one in the resampling step. Of all the steps of the algorithm, the sensor update is by far the most computationally intensive and its acceleration is therefore the focus of this work.

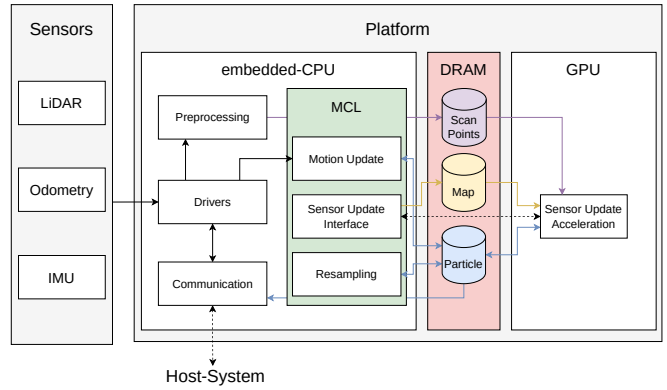

Fig. 3: Visualization of the system architecture for the GPU.

IV. System Architecture

In this section we describe the basic system architecture for the CPU and GPU implementation of the algorithm, where the CPU-based implementation serves as the baseline to evaluate the acceleration based on the embedded GPU. To ensure comparability, all systems have the same structure and are able to receive and send data via the same interface, which allows to evaluate them on the same datasets. Moreover, the GPU-based implementation can be considered as an extension of the CPU-based implementation. An overview of the system architecture is shown in Fig. 3. In both cases embedded platforms with CPU and GPU are used. As described in Seq. III, the sensor update is the most computationally expensive part of the algorithm. Therefore, this step is executed on the acceleration hardware, while the motion update and the resampling remain on the embedded CPU. The systems are able to operate without the need for an external computer, but can also communicate with a host using ROS to receive sensor data, visualize the results or replay pre-recorded data via bag files. Odometry estimation for the motion model update is provided by dead reckoning or IMU. 3D point clouds are received from a laser scanner to evaluate the current particles in the TSDF map. On GPU-based platforms, an interface process is running on the CPU to handle the access to the GPU.

V. Implementation

The provided implementations to accelerate the evaluation of the particles on all hardware platforms are detailed in this section. The main algorithm of the sensor update consists of two consecutive steps. First, the sampled poses $p_t^{[n]}$ must be evaluated to compute the weights $w_t^{[n]}$ of the particles. Afterwards, the weights are used to determine the weighted sum of the considered poses to get a pose estimation of the robot system. These steps have to be implemented differently with respect to the underlying hardware architecture.

A. CPU Implementation

For comparability in terms of power consumption and performance, all available threads of the CPU-based system have to be used to achieve the maximum utilization of the

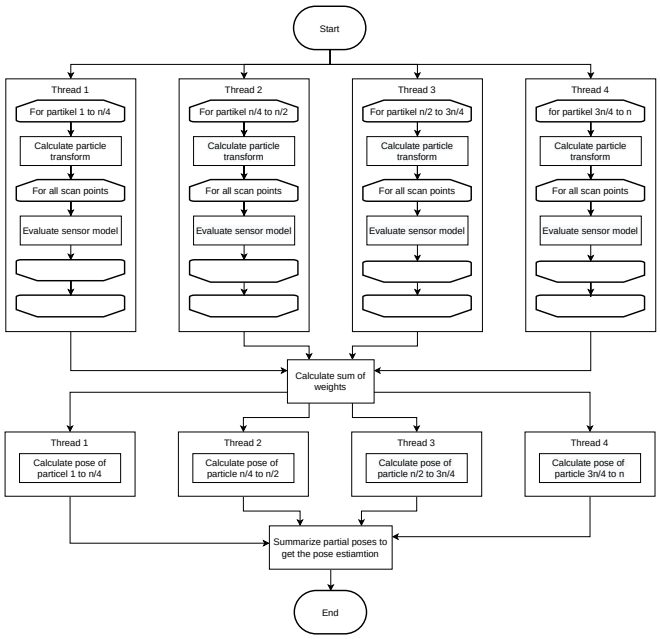

Fig. 4: Program flow of the CPU-based implementation with four available threads. The particles are split up into groups according to the number of available threads.

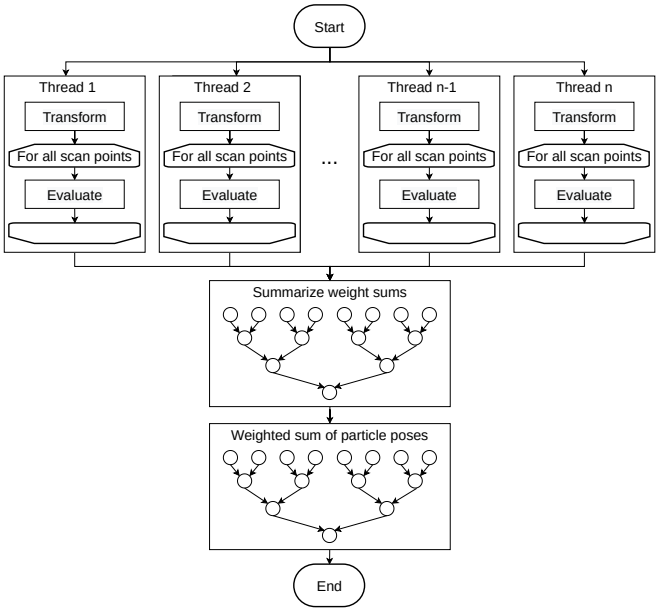

Fig. 5: Program flow of the GPU-based implementation. Each particle is processed by single thread.

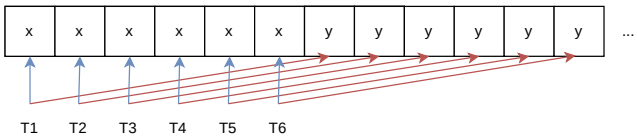

Fig. 6: Visualization of the access pattern for the particles and the scan points per thread on the GPU. Every letter stands for one coordinate of the particles or the scan point.

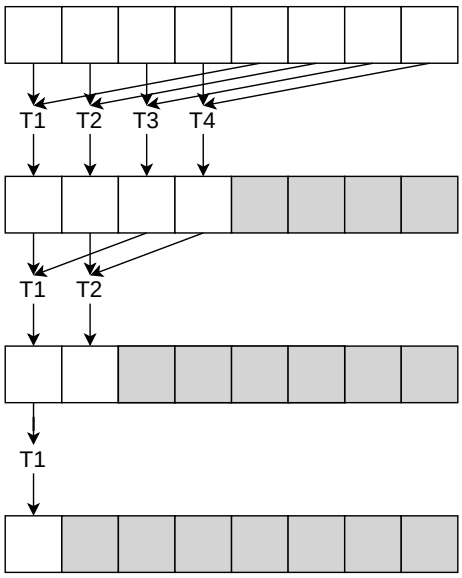

Fig. 7: Simplified example of the reduction pattern used to merge the partial results on the GPU. Each thread summarizes two data in every iteration until only one final data point is left. The gray fields mark data points no longer considered by the algorithm.

hardware. The program flow of the implementation is shown in Fig. 4. The particles are split up into groups according to the number of available threads in the system. Every group of particles is processed in parallel by an assigned single thread. After evaluation of all particles, the threads are synchronized to sum up the particle weights, which is required for normalization. The calculation of the pose estimation is also split up into the available threads similar to the evaluation procedure.

B. GPU Implementation

For maximum throughput on GPUs, many independent threads have to be scheduled to hide latency inside the system. The idea is to adapt the CPU-based implementation from the previous section by processing every particle in a single GPU thread. This is visualized in Fig. 5. Despite the property, that all particles can be evaluated independently, one major bottleneck to be dealt with is the high amount of memory access on the particles, scan points and map data. Although the map access is highly unstructured and strongly depends on the distribution of the particle cloud, the access to the particles and scan points in the algorithm is structured. Hence, the memory efficiency can be optimized by ordering the data according to the threads in the multiprocessors, as visualized in Fig. 6. Although this leads to higher parallelism, it also increases the synchronization overhead for computing the particle weights and the pose. This is handled by implementing a GPU-based reduction pattern, which exploits the memory hierarchy of the GPU. Depending on the number of used particles in MCL, the reduction kernel needs to be called several times to achieve synchronization of all threads. This is because of the processing structure in the GPU, which

allows synchronization between threads only within a thread block of limited size. So every iteration of the reduction pattern summarizes the thread blocks hierarchically to exploit parallelism, as shown in Fig. 7. In every iteration, each thread sums up two data points. Thus, in every iteration the number of data points and active threads is halved until the final result is reached. Similar to the particle ordering, the threads access and store data in the same order as they are executed on the multiprocessors of the GPU, resulting in a better utilization of the memory bandwidth.

VI. Evaluation

To evaluate the implementations for the CPU and the GPU we used pre-recorded datasets. First, we used an office floor, shown in Fig. 8, to test our implementation in a common GPS-free indoor environment. It covers an area of $50m \times 20m$. In addition, a robot equipped with a virtual Velodyne VLP-16 LiDAR was simulated in this environment using Gazebo. In the original Gazebo environment a safely driving robots moves on the xy-plane only. To demonstrate the localization of a robot moving in 6D state space, we placed ramps into the map as shown in the left part of Fig. 8. Using a simulation environment lets us investigate an arbitrary number of motion trajectories within the same map where each motion trajectory is provided with a ground truth localization.

Second, we used datasets of the HILTI SLAM Challenge [22] to especially evaluate the GPU-based implementation to its capability to globally localize a robot in real-world indoor scenarios. We focused on the "Drone-Testing arena" sequence because it contains sensor data acquired from an Ouster OS0-64 and, most importantly, provides a 6-DoF trajectory of the sensor as ground truth. We first use the ground truth as a perfect odometry estimation for the motion update to avoid false odometry estimations affecting the global localization. From there on, we could systematically increase the level of applied noise which allows us to make accurate predictions about how much odometry estimation error is allowed in order for our system to still produce reliable global localization results.

For both the simulated and the real-world experiments we generated the required TSDF maps using HATSDF-SLAM [23]. Then, the sensor data and the respective TSDF map is passed into our software via the provided ROS interface to ensure comparability. During the following experiments we tested our implementation for a large number of trajectories, observing whether and how the algorithm converges, and additionally measuring the localization accuracy, performance, and power consumption. To ensure the generalization of our implementation, the evaluation experiments use trajectories different then those used to create the map.

A. Qualitative Evaluation

First, we test our implementation in the simulated office environment. Fig. 1 shows an example experiment. Initially, the particles were uniformly sampled from the free space and with six degrees of freedom. After a few iterations of MCL, the particles are distributed around several probable poses, reflecting symmetries and the similarities of the rooms. Finally, the particle distribution converges around the correct pose of the robot. White dots (best seen in the PDF file) visualize the transformed point cloud of the laser scanner, which, once it matches the map, indicates successful localization. The reconstructed trajectory in this scenario overlaps with the ground truth, as visualized in Fig. 8 (middle). For this run, the translation errors for every axis is plotted in Fig. 10. Repeating the experiment for 11 different trajectories shows that approximately 50,000 particles are needed to achieve robust localization of the robot in this particular environment. Furthermore, both the CPU and GPU implementation achieves a localization accuracy in all simulation experiments within the order of magnitude of the fine map resolution (about 6 cm).

Besides the simulated scenarios, we also evaluated the quality of the global localization in a real-world environment of the HILTI datasets. Similar to the simulated environment, the particles are initially distributed uniformly in the complete free space of the environment. For the Drone-Testing arena, the distribution of the particles for different iterations can be seen in Fig. 9. Similar to the simulated scenario, several particle clusters are formed at the beginning, but converge against the true pose of the robot. We found that increasing the noise of both the velocity and angular velocity components of the motion estimate to $\sigma = 0.1$ does not significantly affect the convergence speed. When setting the noise σ to 0.1, 100,000 particles are needed to achieve a reliable global localization in this scenario. The convergence of the estimated pose can also be seen in Fig. 11, where the translation error for every axis is plotted for the Drone-Testing arena sequence. The results show that the localization error increases at the beginning. This is because of the multiple particle clusters, which lead to a wrong average pose estimation. Then, the pose error decreases significantly as the individual clusters disappear. Overall, the results on the HILTI datasets show that our software is able to perform global localization even in real-world, feature-less environments that contain many symmetries, while moving freely in 6DoF space.

B. Performance

In case of the simulated robot, the laser scanner can provide point clouds with a frequency of 20 Hz. Therefore, the implementations have to compute an iteration of the sensor update in less than 50 ms to achieve real-time performance, while the Ouster laser scanner provides scan points with 10 Hz. Hence, the sensor update must be computed within less than 100 ms to achieve real-time performance in the HILTI datasets. To evaluate the performance, an Intel NUC (NUC6i7KYK, Core i7-6770HQ) served as CPU baseline. For the GPU implementation, we used a Jetson AGX Xavier and a Jetson AGX Orin to accelerate the algorithm and to show the scalability of the implementation. To compare the performance of the implementations, the run time of the sensor update was measured for normally distributed particle clouds with an increasing number of particles. The measured run times are shown in Fig. 12. As expected, the run time

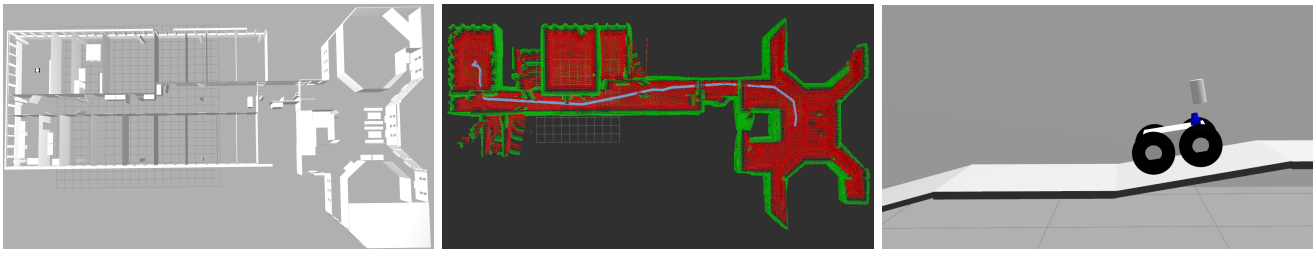

Fig. 8: Top down view of the simulation environment (left), the corresponding TSDF map (middle) and the ramp scenario (right). The traveled path of the robot is marked blue.

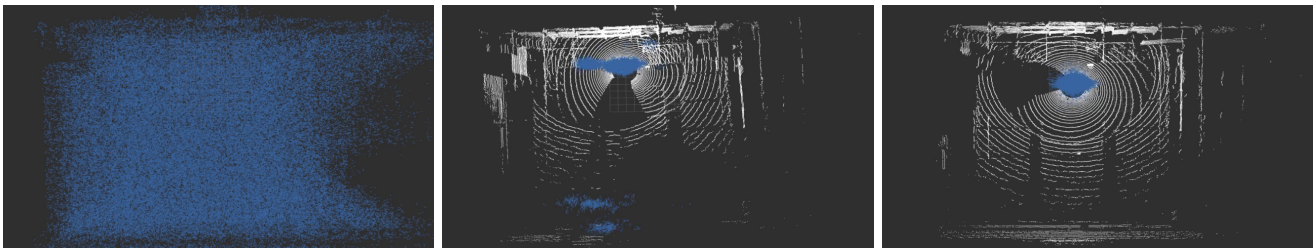

Fig. 9: Top down view of the particles during the initialization (left), after several updates (middle) and near conversion to the real pose (right) while processing HILTI's Drone-Test arena dataset.

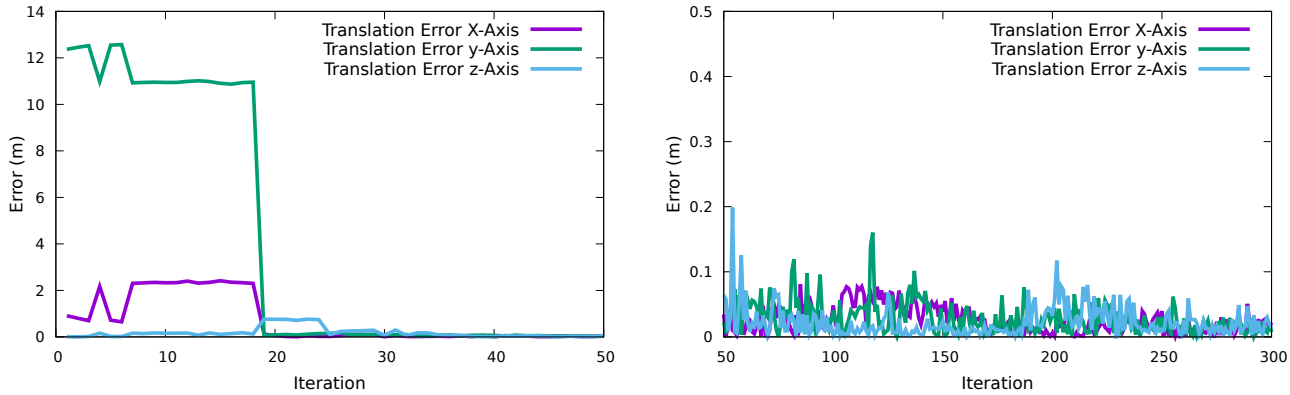

Fig. 10: Translation error for every axis during global pose estimation (left) and the following pose tracking (right) of the using the simulation dataset.

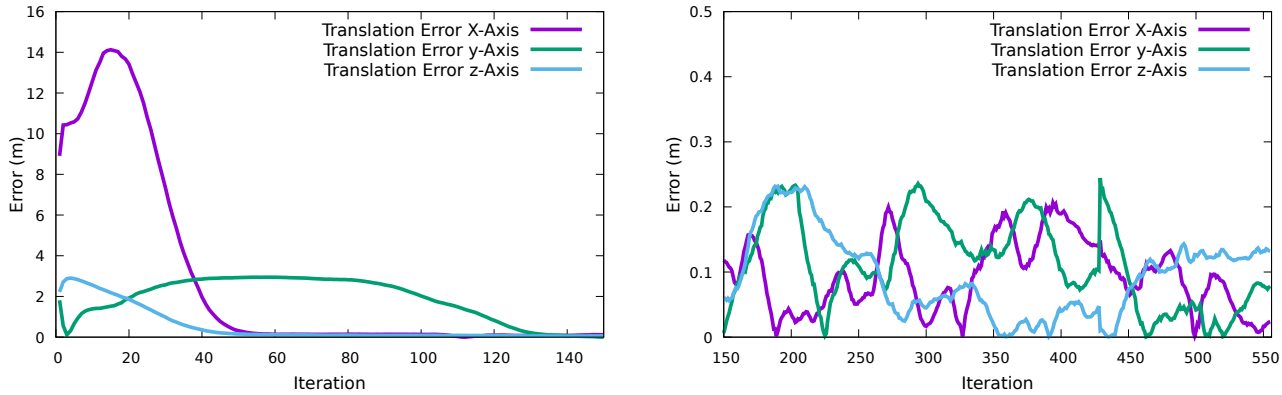

Fig. 11: Translation error for every axis during global pose estimation (left) and the following pose tracking (right) using HILTI's Drone-Test arena dataset.

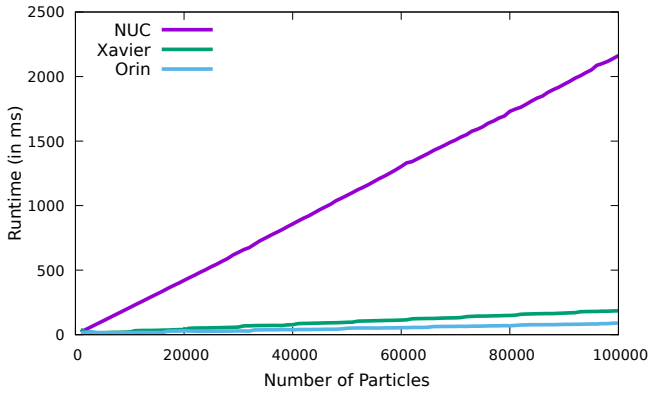

Fig. 12: Comparison of the measured run times with an increasing number of particles during a normal distribution.

TABLE I: Performance and power consumption of the implementations on the CPU (NUC) and GPU (AGX Xavier and Orin) in the simulated environment.

	NUC	AGX Xavier	AGX Orin
Run time [ms]	1527	155	51
Power [W]	72.2	34.64	41.2
Energy [J/Particle Cloud]	110.25	5.37	2.1

of all implementations scales linearly with the number of particles.

In addition, the run times of all implementations were measured during the localization on the traveled path in the simulated environment. Tab. I shows that although the goal of being faster than the maximum scanning frequency of 20 Hz was not achieved, both accelerations based on the GPU are able to speed up the sensor update significantly.

C. CUDA Metrics of the GPU-based Implementation

The GPU-based implementation has been developed using CUDA. So various metrics can be achieved to analyze the efficiency of the implementation on the hardware. They are estimated using a prerecorded normal distributed particle cloud with the tool nvprof. The most important metrices for the particle cloud are depicted in Tab. II. As can be seen, the implementation achieves a high occupancy of the GPU. The remaining resources of the GPU cannot be occupied, which occurs because of the look-ups in TSDF map. The random look-up of a TSDF value can lead to one or two memory accesses to the DRAM, depending on the occupancy of the requested grid cells. So that threads finish their work at different times, leading to inactive resources.

The random distribution of the particles in the environment of the robot has also an impact on the performance and efficiency of the memory access. The look-up of the TSDF values reduces the cache hits significantly. Although some look-ups share the same coarse grid, the most memory access for the transformed scan points are widely distributed in the environment resulting only in a few shared memory access on the fine map between the scan points. The widely and

TABLE II: Considered CUDA metrics for the GPU-based implementation for evaluating a locally initialized cloud of 50,000 particles.

Metric	Value
Achieved Occupancy	74,163 %
L1/TEX Cache Hits	39,64 %
L2 Cache Hits (read)	39,63 %
Global Memory Efficiency (read)	46,43 %
Global Memory Efficiency (write)	100,00 %

randomly distributed particles also lead to unaligned memory access in the TSDF map between neighboring threads. This reduces the efficiency of the read accesses, because only a few parts of the read data blocks read from DRAM can be used. Furthermore, the cache hits decreases as the particle cloud is distributed more widely in the environment. When the particles are initialized globally the L1 cache hits decreases to 31.53 % and the L2 cache hits achieved a value of 11.47 % However, the writing accesses into DRAM can be performed with the maximum efficiency, because all threads write the weights of the particles aligned back into the global memory. All in all it can be concluded, that the memory accesses to look up the TSDF value in the map of the environment are the major bottleneck for the implementation on the GPU, because the underlying hardware architecture does not fit the kind of reading accesses.

D. Energy Efficiency

For localization on a mobile system, power consumption is a crucial factor. We measured the power consumption via a shunt directly on the boards. The results are shown in Tab. I. With the GPU implementations, a significant reduction in both power and energy has been achieved. Using our implementation on the AGX Orin resulted in a more than 50 times decrease in Energy per Particle Cloud compared to the CPU implementation.

VII. Conclusion and Future Work

Because of the significant acceleration and the high energy efficiency the GPU implementation compared to the CPU-based baseline, a decisive step has been made in the direction of a real-time capable global MCL for mobile robots. Our GPU implementation outperforms a CPU implementation by factor of 30, while increasing the energy efficiency significantly by a factor of more than 50. The next step to improve performance and energy efficiency is to build heterogeneous architectures to extend the GPU-based architecture with specialized units to process particles. Furthermore, to evaluate the quality of the invented algorithms in realistic scenarios, new benchmark datasets are required that allow to evaluate many trajectories in challenging environments. To tackle this problem, we plan to provide a repository of reference TSDF maps with many different trajectories captured with a laser tracking system to provide a benchmarking environment for the development of such algorithms similar to the established KITTY [24] and Hilti [25] datasets for SLAM.

REFERENCES

[1] M. Eisoldt, J. Gaal, T. Wiemann, M. Flottmann, M. Rothmann, M. Tassemeier, and M. Porrmann, "A fully integrated system for hardware-accelerated tsdf slam with lidar sensors (hatsdf slam)," *Robotics and Autonomous Systems*, vol. 156, p. 104205, 2022.

[2] S. Rahn, P. Gehricke, C.-L. Petermöller, E. Neumann, P. Schlinge, L. Rabius, H. Termühlen, C. Sieh, M. Tassemeier, T. Wiemann *et al.*, "Redrose—reconfigurable drone setup for resource-efficient slam," in *Proceedings of the DroneSE and RAPIDO: System Engineering for constrained embedded systems*, 2023, pp. 20–30.

[3] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte Carlo Localization: Efficient Position Estimation for Mobile Robots," *AAAI/IAAI*, vol. 1999, no. 343-349, pp. 2–2, 1999.

[4] D. Fox, "KLD-sampling: Adaptive Particle Filters and Mobile Robot Localization," *Advances in Neural Information Processing Systems (NIPS)*, vol. 14, no. 1, pp. 26–32, 2001.

[5] D. Hahnel, W. Burgard, D. Fox, and S. Thrun, "An Efficient FastSLAM Algorithm for Generating Maps of Large-Scale Cyclic Environments from Raw Laser Range Measurements," in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, vol. 1. IEEE, 2003, pp. 206–211.

[6] G. Grisetti, C. Stachniss, and W. Burgard, "Improved Techniques for grid mapping with rao-blackwellized particle filters," *IEEE transactions on Robotics*, vol. 23, no. 1, pp. 34–46, 2007.

[7] F. J. Perez-Grau, F. Caballero, A. Viguria, and A. Ollero, "Multi-sensor three-dimensional Monte Carlo localization for long-term aerial robot navigation," *International Journal of Advanced Robotic Systems*, vol. 14, no. 5, 2017.

[8] S. Oishi, Y. Jeong, R. Kurazume, Y. Iwashita, and T. Hasegawa, "ND voxel localization using large-scale 3D environmental map and RGB-D camera," in *2013 IEEE international conference on robotics and biomimetics (ROBIO)*. IEEE, 2013, pp. 538–545.

[9] A. Hornung, K. M. Wurm, and M. Bennewitz, "Humanoid Robot Localization in Complex Indoor Environments," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2010, pp. 1690–1695.

[10] R. Kümmerle, R. Triebel, P. Pfaff, and W. Burgard, "Monte Carlo Localization in Outdoor Terrains using Multi-Level Surface Maps," *Journal of Field Robotics*, vol. 25, no. 6-7, pp. 346–359, 2008.

[11] M. F. Fallon, H. Johannsson, and J. J. Leonard, "Efficient scene simulation for robust Monte Carlo localization using an RGB-D camera," in *2012 IEEE international conference on robotics and automation*. IEEE, 2012, pp. 1663–1670.

[12] S. Kanai, R. Hatakeyama, and H. Date, "Improvement of 3D Monte Carlo localization using a depth camera and terrestrial laser scanner," *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 40, no. 4, p. 61, 2015.

[13] A. Dhawale, K. S. Shankar, and N. Michael, "Fast Monte-Carlo Localization on Aerial Vehicles using Approximate Continuous Belief Representations," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5851–5859.

[14] F. A. Rahman, R. D. H. Al-Fahsia, and I. Ardiyantoa, "GPU-Accelerated Monte Carlo Localization for Mobile Robot Soccer with Omnidirectional Camera."

[15] M. Bolic, P. M. Djuric, and S. Hong, "Resampling algorithms and architectures for distributed particle filters," *IEEE Transactions on Signal Processing*, vol. 53, no. 7, pp. 2442–2450, 2005.

[16] S. Liu, G. Mingas, and C.-S. Bouganis, "Parallel Resampling for Particle Filters on FPGAs," in *2014 International Conference on Field-Programmable Technology (FPT)*. IEEE, 2014, pp. 191–198.

[17] H. A. Abd El-Halym, I. I. Mahmoud, and S. Habib, "Proposed hardware architectures of particle filter for object tracking," *EURASIP Journal on Advances in Signal Processing*, vol. 2012, no. 1, pp. 1–19, 2012.

[18] A. Krishna, A. van Schaik, and C. S. Thakur, "Source localization using particle filtering on FPGA for robotic navigation with imprecise binary measurement," *arXiv preprint arXiv:2010.11911*, 2020.

[19] J. U. Cho, S. H. Jin, X. Dai Pham, J. W. Jeon, J. E. Byun, and H. Kang, "A Real-Time Object Tracking System Using a Particle Filter," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2006, pp. 2822–2827.

[20] S. Saha, N. K. Bambha, and S. S. Bhattacharyya, "Design and implementation of embedded computer vision systems based on particle filters," *Computer Vision and Image Understanding*, vol. 114, no. 11, pp. 1203–1214, 2010.

[21] N. Akai, T. Hirayama, and H. Murase, "3D Monte Carlo Localization with Efficient Distance Field Representation for Automated Driving in Dynamic Environments," in *2020 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2020, pp. 1859–1866.

[22] M. Helmberger, K. Morin, B. Berner, N. Kumar, D. Wang, Y. Yue, G. Cioffi, and D. Scaramuzza, "The Hilti SLAM Challenge Dataset," 2021.

[23] M. Eisoldt, M. Flottmann, J. Gaal, P. Buschermöhle, S. Hinderink, M. Hillmann, A. Nitschmann, P. Hoffmann, T. Wiemann, and M. Porrmann, "HATSDF SLAM–Hardware-accelerated TSDF SLAM for Reconfigurable SoCs," in *2021 European Conference on Mobile Robots (ECMR)*. IEEE, pp. 1–7.

[24] J. Fritsch, T. Kuehnl, and A. Geiger, "A new performance measure and evaluation benchmark for road detection algorithms," in *International Conference on Intelligent Transportation Systems (ITSC)*, 2013.

[25] L. Zhang, M. Helmberger, L. F. T. Fu, D. Wisth, M. Camurri, D. Scaramuzza, and M. Fallon, "Hilti-oxford dataset: A millimeter-accurate benchmark for simultaneous localization and mapping," *IEEE Robotics and Automation Letters*, vol. 8, no. 1, pp. 408–415, 2022.