A Fully Integrated System for Hardware-accelerated TSDF SLAM with LiDAR Sensors (HATSDF SLAM)

Marc Eisoldt[a], Julian Gaal[a], Thomas Wiemann[a,c,*], Marcel Flottmann[b], Marc Rothmann[b], Marco Tassemeier[b], Mario Porrmann[b]

[a] *Osnabrück University, Autonomous Robotics Group, Osnabrück, Germany*
[b] *Osnabrück University, Computer Engineering Group, Osnabrück, Germany*
[c] *DFKI Laboratory Niedersachsen, Plan Based Robot Control Group, Osnabrück, Germany*

Abstract

Simultaneous Localization and Mapping (SLAM) is one of the fundamental problems in autonomous robotics. Over the years, many approaches to solve this problem for 6D poses and 3D maps based on LiDAR sensors or depth cameras have been proposed. One of the main drawbacks of the solutions found in the literature is the required computational power and corresponding energy consumption. In this paper, we present an approach for LiDAR-based SLAM that maintains a global truncated signed distance function (TSDF) to represent the map. It is implemented on a System-On-Chip (SoC) with an integrated FPGA accelerator. The proposed system is able to track the position of state-of-the-art LiDARs in real time, while maintaining a global TSDF map that can be used to create a polygonal map of the environment. We show that our implementation delivers competitive results compared to state-of-the-art algorithms while drastically reducing the power consumption compared to classical CPU or GPU-based methods.

*Corresponding author
Email addresses: meisoldt@uos.de (Marc Eisoldt), gjulian@uos.de (Julian Gaal), twiemann@uos.de (Thomas Wiemann), mflottmann@uos.de (Marcel Flottmann), mrothmann@uos.de (Marc Rothmann), mtassemeier@uos.de (Marco Tassemeier), mporrmann@uos.de (Mario Porrmann)

1. Introduction

Over the years the solution of the Simultaneous Localization and Mapping (SLAM) problem has drawn significant attention in the robotics community. Solving SLAM requires tracking the position of a system while simultaneously building a map that supports self-localization. Since the problem formulation itself is independent from the used sensors and map representation, many different lines of research in this context have been established, ranging from monocular SLAM using RGB cameras [1] to the use of 3D data from LiDARs or other 3D sensors [2]. Often, the SLAM problem is divided into the sub-problems of incremental online SLAM, that aims to integrate the incoming sensor data into the current map in real time, and graph-based SLAM that tries to refine pose estimations in an offline post-processing step. Incremental SLAM is prone to drift due to possible miss-alignments of incoming data. Hence, the consistent integration of all incoming data in real time is required to minimize this drift.

In this paper, we present an approach called HATSDF (Hardware Accelerated TSDF SLAM) for incremental SLAM using 3D LiDAR sensors. This article is an extension of the algorithmic work presented at ECMR2021 [3] and the system design proposed at FPT2021 [4]. It summarizes the implemented algorithms and describes the hardware design and the development process in detail. To evaluate the scalability on the given hardware in terms is scanning resolution, we performed additional experiments with two different LiDAR sensors, namely a Velodyne VLP-16 and an Ouster OS1-128. We show that HATSDF SLAM is able to integrate the incoming data of LiDAR sensors in real time to maintain a 3D environment map[1]. The internal map is represented as a Truncated Signed Distance Function (TSDF). The main benefit of using such a representation is that it can be used to efficiently compute a polygonal mesh, which can be used for different purposes like visual inspection, path planning [5] or as environment representation in simulators [6]. The proposed

[1]The source code is available here: `https://github.com/uos/hatsdf_slam`

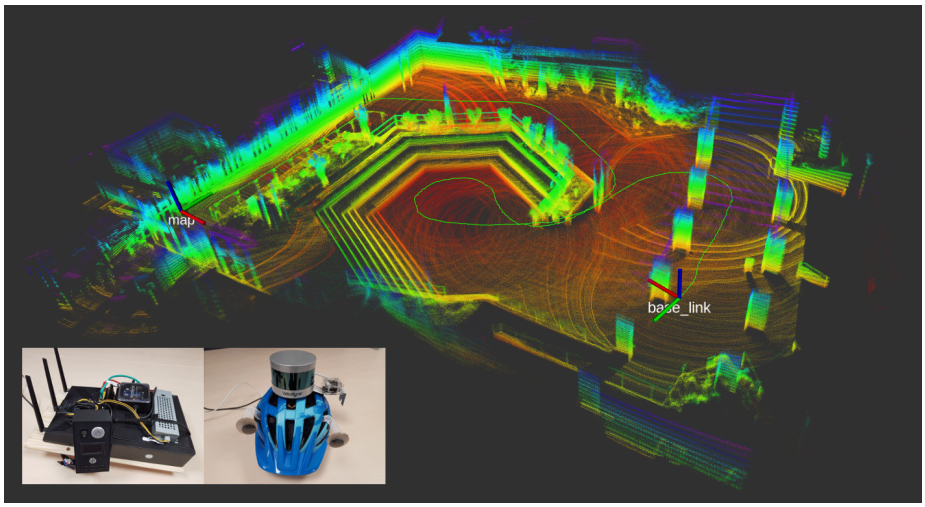


Figure 1: The HATSDF hardware consisting of a head-mounted Velodyne VLP-16 LiDAR and mobile embedded computing system (reconfigurable SoC). The TSDF-based registration to generate the map of an urban environment was done online and in real time using the SoC's CPUs and FPGA. The overall power consumption was significantly lower compared to a regular computer.

algorithm is designed to run on reconfigurable Systems-on-Chips (SoCs) which combine a classic CPU with a Field Programmable Gate Array (FPGA) that allows implementing algorithms in hardware. Such SoCs are extremely energy efficient and are hence apt candidates for deployment on mobile robots.

The proposed system is realized as a stand-alone SLAM box with an optional ROS interface that can input scans from different LiDARs directly on the SoC-System. In addition to this live data, pre-recorded data streams from ROS bag files can be fed into the system for parameter tuning and benchmarking. To demonstrate its accuracy and flexibility, we evaluate the system on new data sets recorded online in indoor and outdoor scenes as well as publicly available reference data sets. The results show that HATSDF SLAM features minimal drift in these scenarios and compares well to established SLAM algorithms, while reducing the required energy per laser frame by a factor of 18 compared to a software implementation on a mobile computer.

2. Related Work

Depending on the application requirements, a wide variety of SLAM algorithms has been developed, significantly differing in computational complexity. Incremental SLAM with 3D point clouds is usually solved with some version of the well-known ICP algorithm [7], that computes the transformation matrix between sets of corresponding points in two point clouds. An often-used extension is the Generalized ICP algorithm [8], which introduces an error function that minimizes the co-variances between sets of corresponding points. Finding the correct correspondences is the key to success in ICP-based algorithms. The original ICP algorithm uses a simple closest point heuristic. More sophisticated methods include Point-to-Plane [9] or Point-Mesh-Correspondences [10]. These methods are able to enhance the quality of the computed transformation matrices, but rely on additional information that has to be derived from the input data. The Point-to-Plane ICP algorithm relies on surface normal estimations, the Point-to-Mesh ICP variant requires some kind of triangle mesh, respectively, increasing the needed computational time. These methods usually assume a stop-and-go scan pattern. Scanning with moving vehicles adds additional problems, as the movement may add additional errors. Hence, methods like LOAM (LiDAR Odometry and Mapping) [11] have been developed for that special purpose. The well-known LeGO-LOAM approach [12] extracts planar features from the point clouds to solve the matching problem.

The main drawback of all of these approaches is that the resulting maps are aligned point clouds or sets of local meshes which are not well suited as robotic maps due to the large memory footprint and missing connectivity of the map elements. For many robotic applications, it would be beneficial to have a closed surface description of the scanned environments. These limitations can be overcome by so-called TSDFs (Truncated Signed Distance Functions). A TSDF is usually represented as a 3D voxel grid, where each vertex of the grid stores the closest signed distance to the nearest surface. The sign encodes the relative orientation while the truncation prevents inconsistencies in convex environments.

The main advantage of such a representation is that distance values between voxels can be interpolated easily, resulting in a pseudo-continuous representation that allows easy extraction of polygonal models using the well-known Marching Cubes algorithm [13]. In the context of SLAM, KinectFusion [14] was the first method to provide such a representation for small volumes by exploiting the structural advantages of RGB-D images in a so-called Projective ICP implemented on a GPU. The initial version was only able to cover a small volume. Kintinuous [15] extended that to larger environments using a clever swapping strategy. Nießner et al. presented an alternative map representation using a spatial hashing strategy to achieve both a memory and speed efficient map representation which suitable for large and fine scale volumetric reconstruction [16]. These algorithms are tailored for RGB-D or time-of-flight cameras, which usually have a limited range and are sensitive to ambient light. In addition, the requirement of a GPU increases the power consumption significantly.

To reduce power consumption in mobile autonomous systems, FPGAs are apt candidates. Hence, several approaches have been proposed to use them in robotic applications. However, hardware acceleration for LiDAR-focused methods is rarely seen in the scientific literature. In TSDF-based SLAM, on the other side, some individual parts of the required processing pipeline have been implemented in hardware. Gautier et al. [17] used OpenCL to implement ICP and Volume Integration on an FPGA. The other parts of the KinectFusion algorithm have not been implemented on the FPGA due to memory bandwidth restrictions. Kosuge et al. [18] improved the performance of picking robots by implementing an FPGA-based ICP accelerator. They achieved more than 11 times faster pose estimation compared to an implementation on a 4-core CPU. Additionally, they leverage partial reconfiguration of the FPGA to save resources by switching the FPGA configuration between graph generation and nearest neighbor search. In addition to these ICP accelerators, Gautier et al. [19] performed a design-space exploration of FPGA-based dense SLAM architectures and implemented an end-to-end dense SLAM accelerator on a Cyclone V FPGA SoC, achieving a throughput of 2 FPS, which is much too low for prac-

tical applications. However, compared to the ARM processor of the SoC, they achieve twice the throughput for the whole design and up to 38 times increased speed for individual algorithms of the system.

By utilizing only a few landmarks or features, sparse SLAM algorithms significantly reduce the computational demands at the cost of localization accuracy. A popular algorithm for sparse SLAM is the extended Kalman Filter (EKF), which has been used, e.g., for an FPGA implementation achieving an update frequency of 44.39 Hz with 20 feature points in a 3D example [20]. In contrast to sparse SLAM, semi-dense SLAM algorithms aim to use a larger, high-quality subset of sensory information. Boikos and Buaganis [21] implemented an accelerator for Large-Scale Direct Monocular SLAM, an algorithm that directly operates on pixel values of camera images. For 320x240 camera images, their FPGA implementation achieves 4 FPS, which is twice as fast as the CPU-based reference implementation. Since DDR memory access is the main bottleneck of the design, Boikos and Buaganis improved the implementation using a dataflow architecture, which achieved a throughput of 22 FPS on a Zynq-7020 SoC [22] and 60 FPS at a resolution of 640x480 pixels using a Zynq-7045 SoC [23].

In this paper, we focus on Point-to-TSDF registration with Velodyne VLP-16 and Ouster OS1-128 LiDARs. Both provide a 360° horizontal field of view combined with a large maximum measurement distance. The used sensors provide up to 128 scan lines, which is significantly more than the sensors used in the previous work discussed above. In our work, we target a completely embedded system solution that is capable of processing the point clouds in real-time with the maximum scanner frequency and full resolution. For that, we present an incremental FPGA-based TSDF SLAM approach implemented on an integrated System on Chip (SoC) to create large-scale TSDF maps from high resolution LiDARs. It is neither dependent on odometry nor GPS and shows only low drift due to two factors: First, it is able to process point clouds of the LiDARs in real time. Second, we use a Point-to-TSDF registration method [24] that is known to be precise, but computationally expensive on CPUs, which makes an efficient deployment on mobile systems difficult, due to hardware costs and

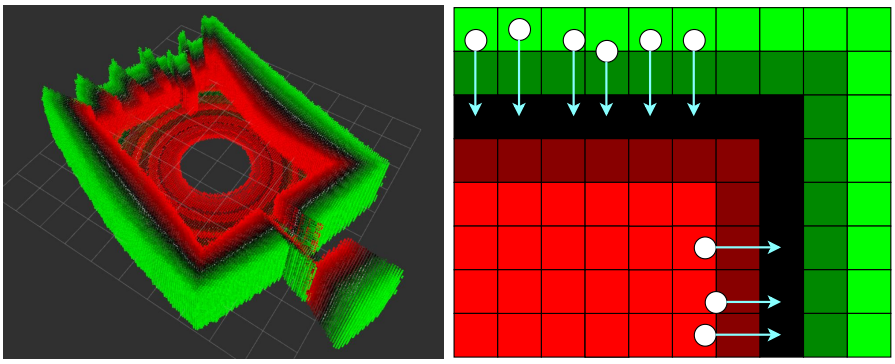


Figure 2: Example of a TSDF voxel grid derived from a single laser scan (left) and an illustration of the Point-to-TSDF approach used in this paper (right). Green voxels represent positive TSDF values, red voxels negative ones. The turquoise arrows visualize the transformation, which is aspired by the registration to match the given set of points shown as white points.

energy consumption

Our completely integrated SoC solution with FPGA acceleration overcomes these drawbacks. We provide a reference implementation and benchmark our system with different sensor resolutions. This so-called SLAM box consists of a small computer case with batteries, SoC and head-mounted laser scanner. Fig. 1 shows the system and an example map. To prove its energy efficiency, we compared the resulting power consumption with a software implementation running on a mobile computer. We assessed its precision in online experiments using the integrated standalone sensor system ("SLAM box") while walking and offline comparison with publicly available data sets.

3. TSDF SLAM

There are many approaches to solve the Simultaneous Localization and Mapping problem. In our work, we use an incremental TSDF-based algorithm that integrates the LiDAR data into a global TSDF map representation based on pose estimations from an IMU. Our approach transfers the GPU-based swapping strategy of Kintinuous [15] to an FPGA accelerated version. Individual map parts, so called local maps, are calculated from a set of several scan frames.

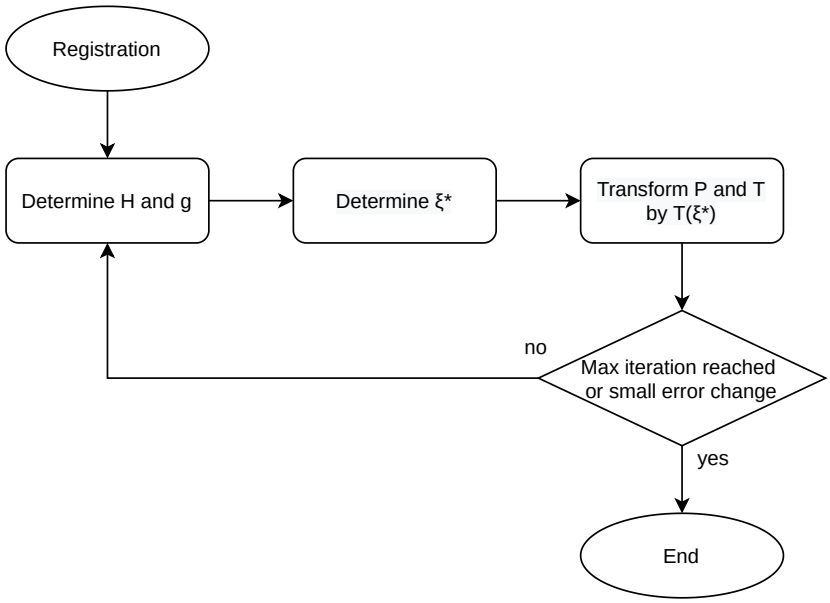


Figure 3: Flow diagram of the iterative Point-to-TSDF registration procedure

If the spatial region of a local map exceeds a predefined maximum, it is fused into a global map representation after successful registration. This approach allows the system to map large environments without the need for a more complex resource-intensive implementation of sparse data structures to represent the scanned environment as suggested in [16]. The global map is stored persistently on the disc in a Hierarchical Data Format (HDF5). HDF5 is a commonly used data storage format that allows to save and load multi-dimensional arrays efficiently. It allows to organize the data in a custom schema. The main benefit of using HDF5 is that all data is stored in a single file. Our previous evaluation [25] proved that robotic maps stored in this format can be accessed faster compared to direct reading from single files. Within the HDF5 file the local TSDF maps are stored as single data sets. The indices of the local chunks within the global map are encoded in the meta data fields of these data sets. Using HDF5 in our system provides a fast access to the required data around the current position and supports the implemented swapping strategy, such that the local map can be updated very efficiently with respect to the changing position

of the SLAM-Box.

A Truncated Signed Distance Field (TSDF) is a 3D voxel array providing an implicit surface representation. Each voxel is labeled with the distance to the nearest surface. Positive values indicate free space, zeroes represent the surface, and negative values represent occupied areas behind the surface. For generating a TSDF volume representing the LiDAR data, the TSDF values are generated based on a method similar to the one described in [24]. For each scan point, the corresponding ray is traversed, and at every intersection with a voxel, the truncated distance to the scan point is taken as a new entry. Furthermore, the new calculated TSDF values are integrated into the current map by a running average mechanism. For this purpose, a weight for every map entry is stored, which represents its certainty. In addition, weights are calculated depending on the distances to the scan points. Depending on the voxel size, a grid cell can intersect multiple rays. Only the map entry, representing the smallest distance to the scanned surface, is used for the map update. This is why the calculated values must be stored temporarily before they can be fused with the current map. Furthermore, the scan points are ordered in horizontal scan lines, each recorded with a different opening angle. This leads to gaps in the TSDF volume, which disrupt the localization. Therefore, the TSDF values are interpolated orthogonal to the scan rays to fill the gaps as long as no other non-interpolated entries are determined. To register the incoming frames with the local map, we use a Point-to-TSDF approach [24] to determine the transformation between successive poses, where a newly acquired scan is registered using the most recent local map. This task can be described as a minimization problem as shown in Eq. 1.

$$\xi^* = \arg\min_\xi \sum_{i=1}^{|P|} \|D(T(\xi) \cdot p_i)\|_2^2 \tag{1}$$

$$\xi = \begin{bmatrix} \omega_1 & \omega_2 & \omega_3 & v_1 & v_2 & v_3 \end{bmatrix}^T \tag{2}$$

P denotes the current set of scan points, while D contains the signed distance values for every point to the reference map. As shown in Eq. 2, ξ is

a six-dimensional vector representing the motion between two frames as a rotational velocity ω and a linear velocity v for every axis and $T(\xi)$ returns a transformation matrix based on this representation. Since the registration error can be considered as the distance of the scan points to the actual scanned surface, the deviation can also be calculated as the sum of the TSDF values of the points itself. The solving strategy for this optimization problem is to move all scan points in an iterative manner in the direction of the decreasing distance values until the surface is reached, while the IMU data is used as an initial rotation estimation. The idea is illustrated in the right part of Fig. 2 on a 2D example. In each iteration, an intermediate transformation is determined based on every scan point, which is used to move the points further in the direction of the surface until they are close enough to the surface cells or a maximum number of iterations is reached. Eq. 3 describes how an intermediate transform is built, and Eq. 4 and 5 describe how the matrices H and g can be determined based on all received scan points.

$$\xi^* = -H^{-1} \cdot g \tag{3}$$

$$H = \sum_{i=1}^{|P|} J(p_i) \cdot J(p_i)^T \tag{4}$$

$$g = \sum_{i=1}^{|P|} J(p_i) \cdot D(p_i) \tag{5}$$

A crucial part for solving this problem is the Jacobian matrix, which is calculated based on the TSDF neighborhood around every point as shown in Eq. 6. According to the minimization problem, the matrix represents the gradient of the distance field with respect to ξ and can be determined by applying the chain-rule, where the TSDF gradient on the right side points in the direction of the surface and can be calculated for the discretized TSDF map using a central

differential quotient.

$$J(x) = \nabla_\xi D(x) = \frac{\delta x}{\delta \xi} \frac{\delta D(x)}{\delta x} = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \frac{\delta D(p)}{\delta x} \\ \frac{\delta D(p)}{\delta y} \\ \frac{\delta D(p)}{\delta z} \end{bmatrix} \tag{6}$$

In addition, the changing rate involves a damping factor, which is linearly increasing with each iteration while the impact of every scan point can be weighted. The total transformation between two consecutive scan frames is the combination of the intermediate ones. The complete iterative procedure is summarized in Fig. 3. Both the mapping and the localization procedure show a high potential for parallelization as provided be the reconfigurable hardware architecture of the used SoC. The next sections describe the specific FPGA implementation and its corresponding system architecture.

4. FPGA Implementation

In this section, the implementation of the SLAM algorithm on the reconfigurable SoC is detailed. First, an overview of the accelerated algorithm is provided and the hardware-software partitioning is discussed. This is followed by a more detailed explanation of the developed FPGA kernels that are used in the localization and mapping procedures.

4.1. Pipelining of the Algorithm

HATSDF SLAM mainly consists of three processing steps: pre-processing, registration, and TSDF map update. The pre-processing step applies different filters to the raw input point cloud. First, a ringwise median filter is applied to remove outliers. In order to improve the overall runtime of our system, a reduction filter is applied to significantly reduce the number of points in the point cloud while simultaneously keeping enough information to obtain good

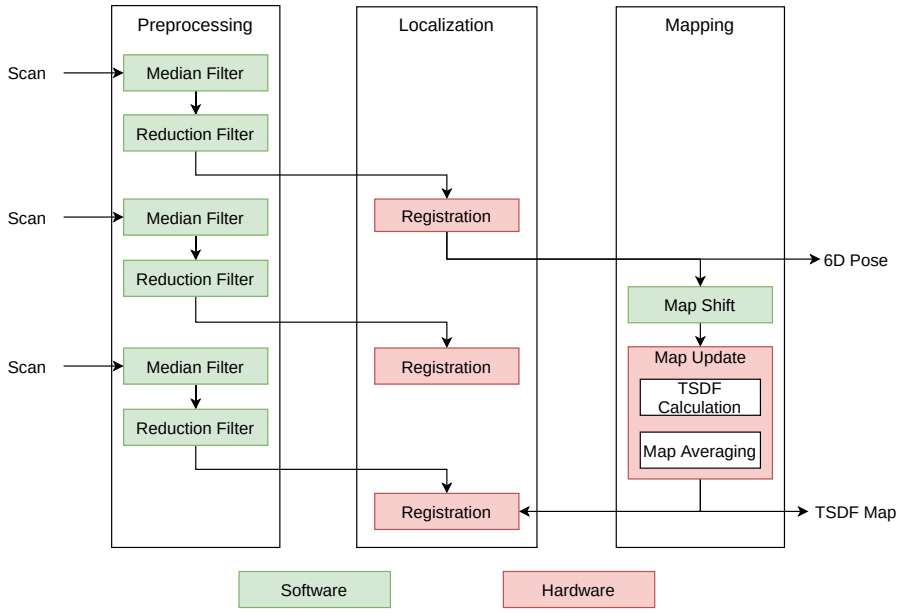


Figure 4: Schematic data flow in HATSDF SLAM. Preprocessing, registration and TSDF map update are pipelined to maximize data throughput.

results in the registration step. The last task of the pre-processing step involves the accumulation of IMU data published between the last and current scan to get an initial rotation estimation. The other two processing steps, registration and TSDF map update, have been discussed above.

Since each processing step is dependent on the results of the previous calculations, the capabilities for parallel computations for a single scan are limited. Despite this limitation, we were able to greatly improve the data throughput of our system by using a pipelining approach, as shown in Fig. 4. For this, each processing step is executed in its own thread. This allows new scans to be processed as soon as it becomes available instead of having to wait for the previous scan to be processed completely.

The overview of the architecture in Fig. 5 shows the kernels in the FPGA with their internal structure and their connections to the external DRAM memory. Additionally, it illustrates the connection to the Processing System with the embedded Arm CPUs and the interfaces to the periphery as well as the

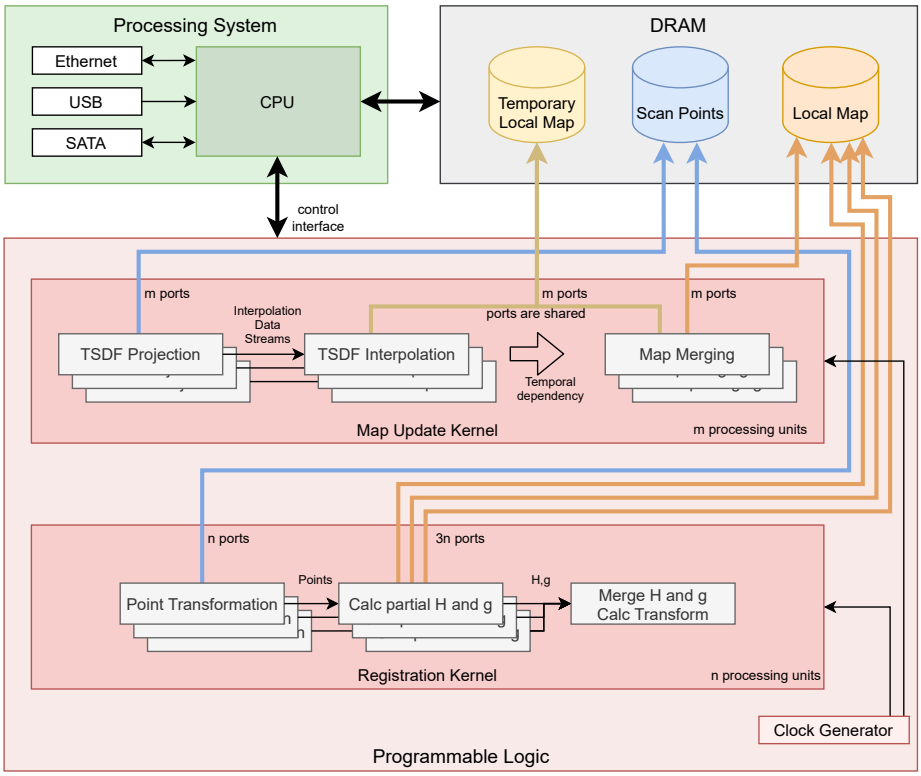


Figure 5: High-level architecture of the SLAM implementation

different clock domains that are used to maximize the overall throughput. The main bottleneck of the implementation is the access to the local map, located in the DDR4 memory. Although the local map only represents a small fraction of the global map, it is too large to fit into the internal memory resources of the FPGA. Depending on the scanned environment, the size of the local map ranges from 20 to 200 MB.

4.2. Storing the map

The TSDF map is represented as a 3D grid with TSDF values and weights for every voxel. Since large maps would utilize more than the available main memory, our algorithm only uses a local map around the current pose for registration. As described in [26], the global map is represented as a set of so called

"chunks" of a fixed cubic size that can be accessed on demand to replace the local map, if the detected pose offset exceeds a given maximum. The chunks are spatially indexed according to their position in the global voxel grid and stored as serialized arrays in dedicated datasets in a HDF5-file. The name of each dataset represents the position of the chunk in the global map. Besides the local map our application also caches a set of chunks around the local pose in main memory to speed up loading the relevant data. After mapping, the HDF5 file containing the global map can be downloaded from the SSD of the HATSDF board. After download, a Marching Cubes implementation can be used to compute a triangle mesh, which is post-processed with Laplacian smoothing [27] and a hole filling algorithm to remove artifacts.

4.3. Implementation of the map update

Simply pipelining the complete implementation does not lead to the performance required to process the incoming data with the maximum scanner frequency of 20 Hz in real-time. Therefore, we modified the execution order to achieve better throughput. The map update takes much longer than the registration, as the former requires a larger amount of slow memory accesses. Assuming that the pose changes only slightly between two scans, an immediate map update has little information gain and thus little effect on the localization. Hence, it is possible to register a few scans before triggering a new map update with the current scan data and synchronize the map afterwards. This allows us to decouple the stages and to increase the total throughput of the scan processing because the registration can be performed with a significantly higher clock frequency than the map update.

Depending on the average moving speed of the system, map updates are triggered after a predefined number of scans, ranging from 1 for high speed in an outdoor environment to 15 for low speed, e.g., using a slowly moving robot inside a building. Additionally, map updates are triggered after significant changes of position to ensure correct registration based on newly recognized features in the changed environment. The positional change is measured between the last map

update and the current position computed by the registration. The threshold can be configured and should be low if the environment contains only sparse features, because some features can disappear in the scan while others may reappear. Using this approach, the system is able to process the incoming data with the maximum scanner frequency of 20 Hz in real-time.

4.4. Acceleration of the Registration Kernel and the Map Update

In Fig. 4, the green steps are executed in software on the Arm cores, while the red steps, namely registration and map update, are implemented as hardware kernels on the FPGA. In our software prototype, these were the bottlenecks that limited the scan rate. They are computationally expensive but can be efficiently parallelized, making them the best choice for hardware acceleration.

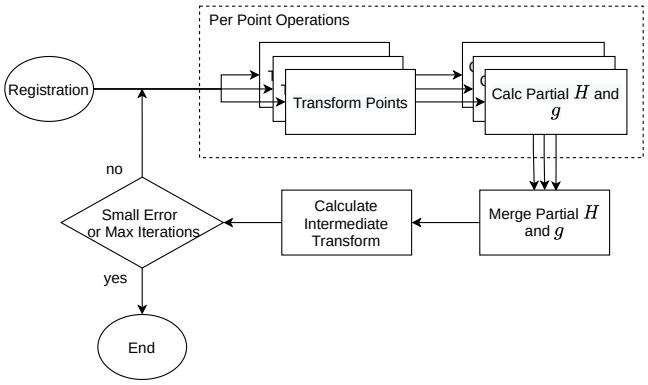


Figure 6: Visualization of the registration dataflow

Fig. 6 illustrates the data flow of the registration kernel. The most computation-intensive steps are the calculations of the H matrix and the g vector (cf. [24]). Since these tasks can be efficiently parallelized, they are executed on multiple processing units. Each unit processes an equally-sized subset of the point cloud. The processing is pipelined so that, ideally, a new point can be processed every clock cycle. This is a kind of instruction level parallelization. Regarding the merging of the H matrices and the g vectors, this leads to the fact that in every clock cycle one partial result from a scan point in added to the total H matrix and g vector. For this reason no further synchronization steps are required

within a processing unit. For the current FPGA system, performance is limited by the available six ports to access the DRAM. Each point requires seven memory accesses to the local map. Based on a design space exploration evaluating different memory configurations, the current implementation uses three units with three memory ports to the local map plus one to the scan points. For smaller or larger FPGAs, the configuration can be easily adapted to maximize efficiency. In the proposed design, a new point can be processed every three clock cycles. Instantiating additional units does not improve performance since the bandwidth of the ports is fully utilized. In the next steps, the partial sums from the H and g calculations are merged. Because three parallel processing units are instantiated, only three partial H and g must be added together. Since this does not result in a bottleneck no additional reduction patterns need to be applied. Finally, the intermediate transformation is calculated (cf. Fig. 6). These steps are sequentially dependent. For minimizing the latency, all loops are unrolled in the high-level synthesis so that operations that are independent of each other can be calculated in parallel.

The TSDF update is divided into two main phases: First, the new distance values and weights are calculated and interpolated based on the sensor data. Then, the results are integrated into the current map, as illustrated in Fig. 5. Phase one is subdivided into two main steps, running in parallel in a pipelined fashion: TSDF projection and interpolation. For each scan point, the ray marching is performed in the TSDF projection step. The algorithm selects the next cell of the TSDF grid along the line between the current position and the scan point. For this cell, the corresponding interpolation area and TSDF value are calculated and inserted into a temporary local map. Finally, the temporary local map is merged with the actual local map. Each of the three processing units (projection, interpolation, and merging) can be instantiated multiple times on the FPGA. Hence, the kernel is easily scalable for different FPGA sizes and memory configurations.

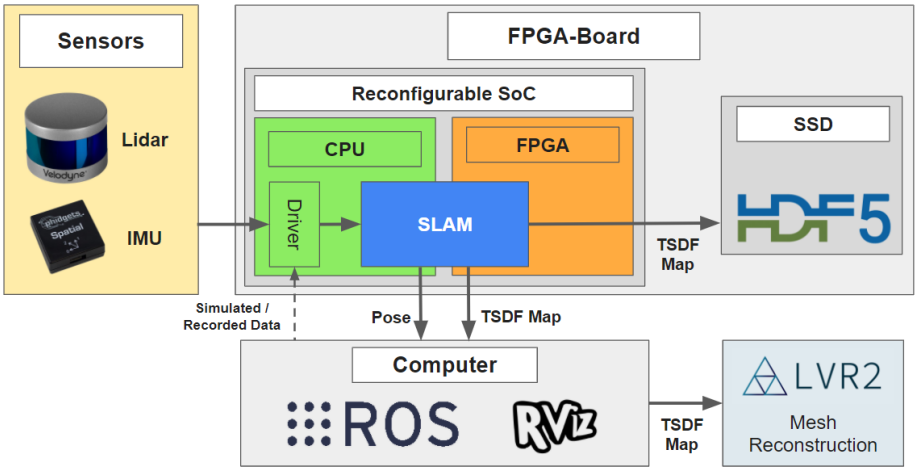


Figure 7: System architecture of the SLAM-Box.

5. System Architecture

The goal of our SLAM-Box is to provide a low power SLAM solution for embedded systems that run on batteries. Therefore, our platform makes use of an FPGA for energy efficient calculations and minimal software overhead to maximize the runtime for a mission. Furthermore, the system is portable in the sense of a dedicated "SLAM sensor" to allow easy integration in different robots and drone systems. Due to the small form factor and low weight of SoC FPGA boards, small aerial vehicles can also be equipped with our system. A standalone scenario is also possible, where the user presses a button to start and stop the mapping. This is used in our evaluation, as our system is placed in a backpack and the sensors are mounted on a helmet.

Fig. 7 shows the system architecture. The SLAM-Box receives data from external sensors, simulated or recorded data. The FPGA board processes these to estimate the current pose and generates the map that is saved to an internal SSD. Via an optional WiFi link, a connected computer system can use the result for further tasks. Additionally, the generated map can be converted to a mesh using LVR2 [26]. The following subsections describe the system architecture in detail.

5.1. FPGA Board

We use a Trenz UltraSOM+ FPGA module equipped with a Xilinx UltraScale+ XCZU15EG SoC, embedded in an UltraITX+ base board the core of our platform. This reconfigurable SoC combines a quad-core Arm Cortex-A53 with a quite large FPGA fabric in a single chip, providing a wide range of opportunities for possible hardware-software partitions to accelerate the application. A SATA-based SSD is integrated for storing the global map and 4 GByte DDR4 memory is available for the local map. Additionally, the hardware platform provides the basic communication infrastructure, including Ethernet (for LiDAR and external communication) und USB (for the IMU). The various tasks have been partitioned between the embedded CPUs and the FPGA, targeting maximum throughput. For the FPGA implementation, a high-level synthesis approach is used, providing a good compromise between development time and algorithm performance.

5.2. Sensor input and output

Our sensor drivers are the main entry points for data from both the LiDAR and IMU sensor. The drivers can be run in two distinct modes: providing live sensor data on the SoC itself or receiving sensor data (live or recorded) from ROS. To take advantage of the flexibility and tooling of ROS, an important but optional element of our software stack is a component for two-way, non-blocking communication between the reconfigurable SoC and an Ubuntu-based Host with ROS installed. This bridge is a ZeroMQ-based messaging library that integrates seamlessly into the ROS ecosystem. LiDAR, IMU, estimated Pose and other debugging data can be sent from the SoC to the host, where it is converted to ROS-compatible message formats.

With this conversion in place, recording data for later use and evaluation is made easy using ROS bag files. In addition, the host can send any data from ROS to the SoC. Central to the bridge is the correct timestamp conversion of the respective systems into their counterpart. These are used for accurate initial rotation estimates in the registration process. While ROS uses a timestamp

in seconds and nanoseconds, our system uses the C++ std::chrono libraries to synchronize using timestamps based on UNIX time. If the bridge is not used, the pose history and resulting map is saved locally on an SSD attached to the SoC, which enables our system to function as an independent unit.

5.3. Hardware-in-the-Loop-based Design Flow

Developing hardware-accelerated algorithms usually results in long development time, dominated by validation and optimization steps. For the SLAM implementation, we used a hardware-in-the-loop (HIL) approach that enabled fast design space exploration together with the ability for fast verification in simulated and real-world environments. The SLAM algorithm was first implemented purely in software to verify basic assumptions made about the feasibility of our approach. This has then been optimized towards a multithreaded reference design on an Intel NUC (NUC6i7KYK, Core i7-6770HQ) and a ROS-based simulated environment has been developed to feed the necessary sensor data into the algorithm. Gazebo was used continuously in all design stages to verify the functionality of the algorithms during parameter tuning and other improvements made to the original prototype.

The simulated environment eases the functional verification of new or changed algorithms, especially since there are no physical sensors required. But since the simulated environment can only provide idealized sensor inputs, depending on the quality of the used models, real sensor data is required for an in-depth evaluation of the algorithm. ROS makes this easy as well since the sensor data generated by Gazebo and sensor data generated by real sensors share the same interface. As real sensor data is typically more prone to noise or to different noise than simulated data, this step was especially important for fine-tuning the parameters.

Both modes of testing and verifying our algorithm mentioned above were crucial for profiling: analyzing the algorithm and finding the parts that can be accelerated by implementing them in hardware. Parts of the algorithm that contribute the most to the overall runtime are extensively analyzed with respect to

possible parallelization since they are predestined for acceleration on the FPGA. For the hardware/software codesign, the Xilinx Vitis tools have been used. The kernels identified for implementation on the FPGA fabric are compiled with the Xilinx High-Level-Synthesis tools based on C/C++ and eventually linked into the final design. On the software side, the Arm cores on the reconfigurable SoC run PetaLinux from which the kernels are called through OpenCL and fed with their respective required data.

After verifying the HLS-generated kernels with simulated data, the system is finally tested with real-world data. In our case, this means that the SLAM-Box is mounted on a mobile robot or strapped to a backpack. For reproducibility, the ROS bridge can be used for recording the live sensor data as well. This data is, e.g., used for comparing map quality and registration performance using different parameters.

6. Evaluation

We evaluated our approach with two LiDAR sensors: A Velodyne VLP16 was used as baseline because sensors with similar specifications were used in related work, e.g., LOAM and the KITTY data sets. The Ouster OS1-128 was used to show how our approach scales to laser scans with higher resolutions and larger field of view. The increased amount of data poses a challenge for our system, since the larger number of points requires more computing resources. Hence, we use this sensor to benchmark the limitations of the proposed system. First, we evaluate the system using the Velodyne VLP-16 to show that real time processing at maximum scanning frequency is possible and delivers consistent maps. In the second evaluation, we aim to increase the quality of the meshes extracted from the internal TSDF representation by using an Ouster OS1-128, which provides up to 8 times more scan lines than the VLP-16. Besides increasing the mesh quality, we use this sensor to evaluate how the increased data size influences the runtime of the registration and TSDF update steps in our algorithm. In a third experiment, we use the ROS interface to compare our

results with established reference data from ROS bag files. Finally, we evaluate the power consumption of the HATSDF SLAM box and compare it with an implementation on a standard computer. To allow reproducibility of the presented results, the descriptions of all system parameters and the configuration files for every considered dataset are provided together with the source code in our Github repository[2].

6.1. Online SLAM in Real Time

To test the system in a structured environment, we scanned an entire floor of an office building (about $500\,m^2$) with the Velodyne VLP-16 and a voxel size of $6.4\,cm$ for the internal TSDF representation. The scanned environment features planar walls, floor and ceilings. The head-mounted LiDAR was moved along the green path shown in the left image of Fig. 8. During the experiment, we visited every room before returning to the starting position. The total length of the recorded path is about $275\,m$. The scanning time was 6:52 minutes and 8 098 scans were inserted into the TSDF map, which was then reconstructed into a mesh using Marching Cubes. To measure the accuracy of the registration and to quantify the global drift, we returned to the exact starting pose marked in the beginning of the experiment. The positional difference between start and end was $7.5\,cm$ and the rotation varied by less than 2^o in Euler angles. This is within the order of magnitude of one voxel in our map. All reported experiments were performed without any external pose estimates.

In addition to the structured office environment, we further evaluated our approach in a less structured outdoor environment. Such environments are generally more challenging due to the lack of floors and ceilings that help to align the data correctly. The chosen environment also featured slopes to show the capability to localize the system in 6 DoF. As our scanner's vertical field of view is only 30 degrees wide, it can only see the ground about $7\,m$ away from the sensor, depending on the height of the person wearing the backpack and helmet.

[2] `https://github.com/uos/hatsdf_slam`

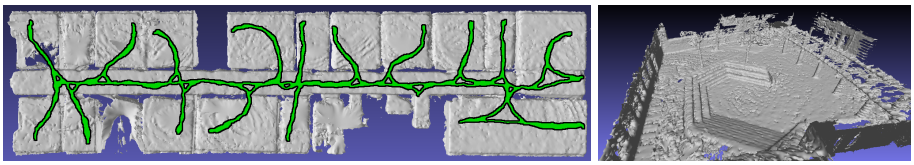


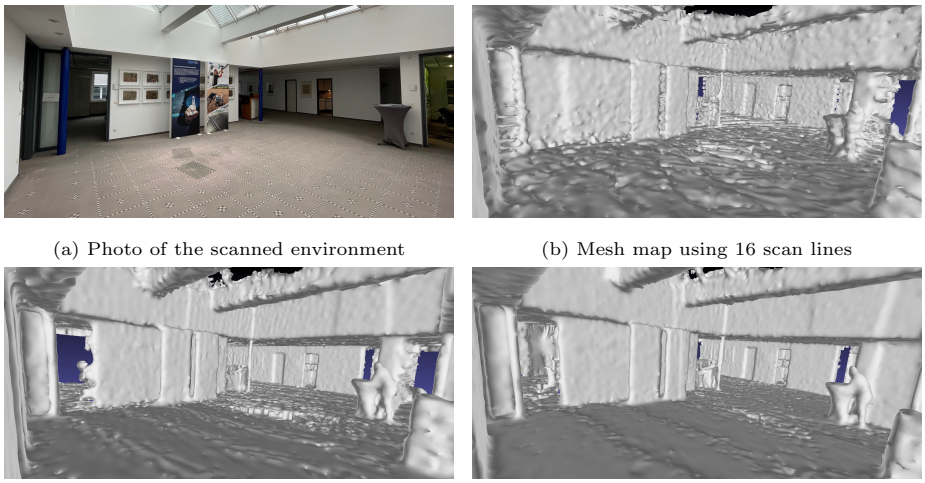
Figure 8: Online evaluation of the HATSDF SLAM in a building interior (left) and an urban outdoor location (right). The images show the reconstructed mesh from the generated TSDF representation. The trajectory in the building data is rendered in green. The indoor data set has a bounding box of approximately 50×30 meters and consists of 8 089 scans. The total path length is $275\,m$. The outdoor data set consists of 3 494 scans. The total path length is $80\,m$.

We therefore needed to reduce the scanner's distance to the ground in order to record enough points on the ground to register with. This was achieved by driving a kart instead of walking. During this second experiment, we recorded a total of 3 494 scans in 2:57 minutes and covered path approximately $80\,m$ in length. The resulting mesh of the map is shown in the right part of Fig. 8. It demonstrates that our approach is able to master height changes, as the ramps present in the chosen scene are clearly visible in the resulting map.

6.2. Online SLAM with an Ouster OS1-128 LiDAR

Using the Velodyne VLP-16 has two inherent issues using our algorithms that were apparent in the previous results: First, the vertical FOV of VLP-16 is on the verge of being too small for a head-mounted LiDAR setup. Depending on the person carrying the LiDAR and the structure of the environment, the ground plane may not be captured by the sensor. Second, 16 scan lines are adequate for precise 3D-registration but fail to produce meshes of high quality due to the large offset between the scan planes, although the HATSDF SLAM algorithm interpolates between consecutive scan planes.

To benchmark the quality of the extracted triangle meshes, we carried out additional experiments with an Ouster OS1-128 LiDAR. This scanner features a 45° vertical field of view and features up to 128 scan lines with a horizontal resolution of up to 2048 points per scan. We chose this sensor for the following



(a) Photo of the scanned environment

(b) Mesh map using 16 scan lines

(c) Mesh map using 64 scan lines

(d) Mesh map using 128 scan lines

Figure 9: Visualization of the triangle meshes extracted from the TSDF map with different number of scan lines. The maps were created using the Ouster OS1-128 LiDAR.

reasons: In general, ICP-based or ICP-adjacent registration algorithms benefit from higher point cloud density in terms of registration precision. Additionally, in our case the algorithm specifically benefits greatly from a higher density in the TSDF map. Finally, the larger vertical field of view of 45 degrees results in a more stable height localization, as more structures on the ground or ceiling are visible due to the larger opening angle. Another benefit for our experiments is that the number of used scan lines can be configured in the sensor's driver. In the presented evaluation, we use this feature to evaluate the scalability of our approach to LiDAR data with higher resolution. For that, we used the ROS-independent driver provided by Ouster[3] which was integrated into our code structure with minimal modifications.

To verify the functionality of the customized Ouster LiDAR driver, we repeated the same experiments originally performed with Velodyne VLP-16 using an increasing number of vertical scan lines, namely 16, 32, 64 and 128

[3]Source code provided on Github `https://github.com/ouster-lidar/ouster_example/`

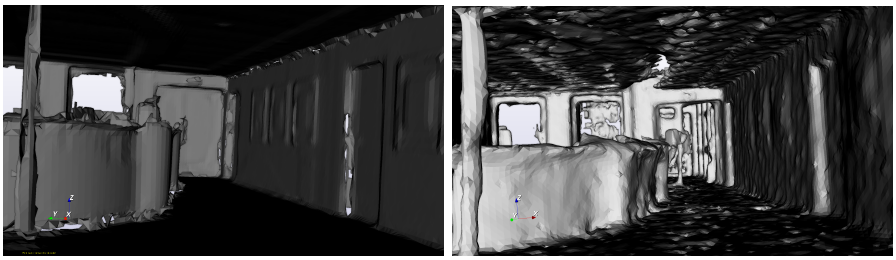


Figure 10: Qualitative comparison between the computed meshes using LVR2 and Riegl LiDAR (left) and HATSDF SLAM with the 128 scan lines from the Ouster OS1.

scan lines, to evaluate registration performance and mapping quality. Figure 9 clearly shows that mesh quality increases as expected with more scan lines. The increased point density together with the extended field of view results in a more accurate reconstruction of the walls, and most notably the floor and ceiling planes. In addition, the contours of objects in the scene are preserved more accurately when more scan lines are used. These experiments lead to the conclusion, that the used mapping implementation has a high potential to accurately reconstruct large indoor and outdoor environments with high quality in short time.

In addition to the HATSDF approach, we scanned the environment shown in Figure 9 with a terrestrial laser scanner (Riegl VZ 400i). The recorded point cloud features approximately 27 million points recorded with a precision of $3, mm$. This high quality point cloud was then reconstructed using the Marching Cubes implementation with signed distance function provided by LVR2 [4] using the same voxel size as the TSDF representation in HATSDF SLAM. Time for reconstruction was 5:45 minutes on an Intel Core i7-4930K with 6 cores and 32 GB RAM and NVIDIA GeForce GTX 1050 graphics card, which was used for normal estimation and signed distance computation. This high quality mesh serves as base line to assess the quality of the maps generated with HATSDF SLAM. Figure 10 compares the resulting mesh from LVR2 with the

[4] Software available here: `www.las-vegas.uos.de`

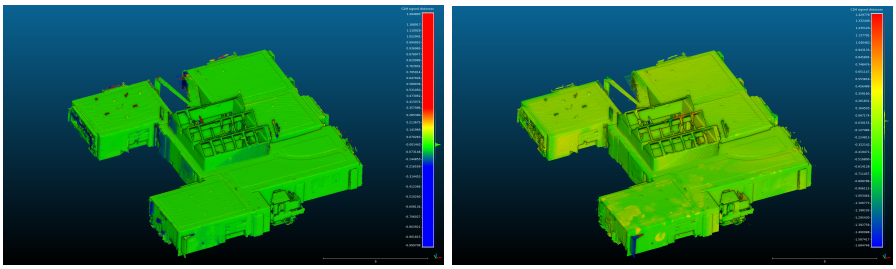


Figure 11: Distribution of the cloud to mesh (C2M) signed distances for the reconstructed mesh with respect to the point cloud recorded with the Riegl LiDAR.

mesh extracted with HATSDF SLAM using the Ouster at full resolution, i.e., 128 scan lines with 2 048 points each. Table 1 compares selected geometric features (measured room sizes, door heights and widths) measured with Meshlab in the triangle meshes. Ground truth was taken with a high precision Leica DISTO Lite 5 distometer (3 mm accuracy).

Additionally, we compared the extracted meshes with the ground truth point cloud from the Riegl LiDAR in CloudCompare [5]. This tool computes the signed distances for all points with respect to a given mesh. A rendering of the comparisons is shown in Fig. 11. The reported error statistics are summarized in Tab. 2. Mean error and standard deviation are derived from a Gaussian distribution that is fitted against the distribution of the cloud to mesh (C2M) errors computed by CloudCompare. The results show that the LVR2 reconstruction approximates the point cloud very well. The mean error is close to zero. The mesh extracted from the HATSDF TSDF volume shows larger deviations. The statistics support the measurements from the manually taken samples in the previous experiments. The computed mean error over all points is about 5 mm for the HATSDF mesh.

The results show that the meshes extracted from HATSDF SLAM reflect the structure of the environment well, although the extracted geometry does not achieve the precision of the meshes reconstructed from the high resolution laser

[5]

Table 1: Comparison of selected geometric features in the meshes computed with LVR2 and HATSDF SLAM with ground truth.

Feature	Ground Truth [mm]	LVR2 [mm]	HATSDF [mm]
Small Door Width	727	730	726
Small Door Height	2420	2456	2306
Large Door Width	812	803	789
Large Door Height	2430	2430	2306
Room Width	6750	6770	6505
Room height	2520	2498	2401

scans. This is due to several reasons. First, our algorithm applies interpolation on the laser data and Laplacian smoothing on the meshes to fill up missing data and remove noise. Especially the Laplacian smoothing removes sharp features in the meshes resulting in deviations from the ground truth. This is reflected in the fact that the reported values from HATSDF SLAM are always lower than ground truth. Second, as described above, the scan matching needs more time with higher number of scan lines which potentially decreases the quality of the TSDF map due to registration errors as discussed more detailed in Section 6.4. Third, the ray marching in the TSDF volume does not provides as accurate signed distance values as the implicit surface reconstructed from the Riegl data due to discretization. Considering these system-inherent error sources, the deviations from ground truth in the computed meshes are small enough to prove that they are suitable for robotic applications, as the measured map errors are usually within the range of some centimeters.

6.3. Offline Evaluation on Reference Data

In order to compare HATSDF SLAM with other approaches, we performed accuracy tests with different LOAM and KITTI odometry data sets. LEGO-LOAM has a similar measurement setup than the one used in the experiments described in Section 6.2. It uses a VLP-16 scanner running at 10 Hz with at-

Table 2: Error statistics for the reconstructed meshes with respect to the point cloud from the Riegl LiDAR as reported by CloudCompare.

	min. dst.	max. dst.	mean	std. dev.
LVR2	0.000	1.170	0.00002	0.024
HATSDF	0.000	1.771	0.00500	0.181

Table 3: Error metrics for the pose estimation of the FPGA-Board measured for the considered KITTI data sets.

Path	ATE [m]	RPE [m]	RPE [°]	Trans. Err. [%]	Rot. Err. [°/m]
6	2.905	0.12	0.21	4.07	1.58
7	7.541	0.14	0.37	5.73	2.82

tached additional IMU. In the considered data set, the robot drives around a building at the Stevens Campus in New Jersey. Since no ground truth is provided, the pose path determined by LEGO-LOAM and the HATSDF SLAM were plotted against each other. The result is shown in the left part of Fig. 12, with the HATSDF SLAM path rotated by 6 degrees to compensate an initial rotational offset. This is because of the discretization of the map representation in which the rotation can only be estimated within the granularity of the TSDF cells. During the replay of the recorded data, HATSDF SLAM showed no significant change in z-direction, although a small slope can be recognized. This is because of the halved scanning frequency compared to our setup. Outdoor scan matching in z-direction strongly depends on features on the ground of the environment. Since the scan lines there are further apart, the density of the TSDF entries is much lower. Also, the system covers a larger distance. Hence, newly captured ground points are located at positions where no TSDF reference is available for registration.

For the KITTI data sets, a Velodyne HDL-64E was used. It records significantly more sensor data and was mounted on a driving car, travelling much

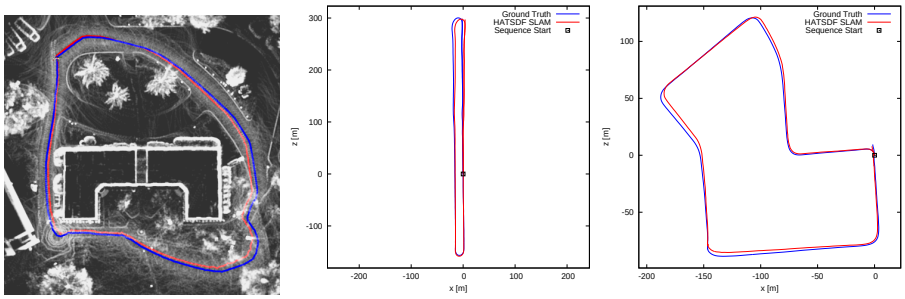


Figure 12: Offline comparison of HATSDF SLAM with reference data sets and trajectories from LeGO-LOAM SLAM [12] (left) and the KITTI data sets 6 (middle) and 7 (right). Pose paths estimated by the HATSDF SLAM are shown in red, reference trajectories are displayed in blue.

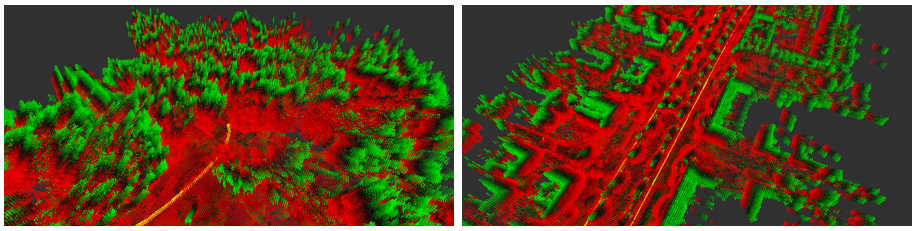


Figure 13: Snapshots of the local TSDF maps recorded during the LOAM path (left) and the KITTI dataset 6 (right). Various outdoor structures such as trees and buildings were mapped during the scanning process.

Table 4: Average translation speed of the robot systems during measured for the considered internal and external evaluation datasets.

	Internal	LOAM	KITTI 6	KITTI 7
Speed [m/s]	0.95	1.30	10.79	6.09

faster compared to our scenarios. Also, no IMU data is provided in the odometry data sets. In contrast to LEGO-LOAM, ground truth for the pose path is available. Due to the larger amount of sensor data per scan frame and larger distances, in this scenario the ICP step in HATSDF SLAM needed more iterations to find a minimum. To compensate for this, we reduced the data rate by a factor of 10 compared to the original data, giving our algorithm time to converge. The estimated trajectories for data set 6 and 7 are visualized in the middle and the right parts of Fig. 12. The error metrics for the calculated trajectories are summarized in Tab. 3.

Although there are strong differences between the setup of the KITTI data set and our scenarios, HATSDF SLAM is still able to provide a good approximation of the reference trajectories. The visible drift occurs mostly at corners due to missing IMU estimates in the KITTI data. As the resources on the FPGA board are limited, we also had to increase the voxel sizes by a factor of three (LEGO-LOAM) and ten (KITTI) to adapt to the extension of local map size to be able to integrate the measured ranges without requiring additional memory. The resulting interpolation error due to higher discretization of the TSDF volume also reduces the registration accuracy. We hope to solve this in future version by implementing a more efficient representation that considers more level of details in the voxel hierarchy. As Tab. 4 shows, the KITTY and LOAM experiments were conducted at higher velocities then in our experiments, which also causes a timing problem on the FPGA, as the clock rate is limited. Given these quite strict limitations of the used hardware, our results reflect the driven path well and strongly indicate that the proposed architecture is applicable in real-life scenarios.

6.4. Resource Requirements and Performance

The resource utilization of the FPGA design is summarized in Table 5. Registration and TSDF kernel contribute most to the total resources. The remainder is required for the base design, including, e.g., the communication infrastructure. The results show that the current implementation achieves a high utilization in terms of the control logic blocks. This is because of multiple instantiations of the hardware and ensures that the SLAM-Box can highly benefit from the resources provided by the used FPGA. It can also be noticed that the implementation has a low total usage of internal block RAM. Hence, the kernels have a high potential to reduce the limiting memory latency, if the block RAM can be used to cache data located in DRAM during the scanning process.

Table 5: Resource utilization of the Xilinx UltraScale+ XCZU15EG FPGA for the registration and TSDF update kernels.

	Registration	TSDF Update	Total
Control Logic Blocks [%]	35.87	46.30	98.95
Look Up Tables (Logic) [%]	23.52	29.63	61.70
Look Up Tables (MEM) [%]	1.71	4.88	15.29
Block-RAM [%]	4.30	2.69	6.99
Digital Signal Processing Units [%]	18.37	18.88	37.25

In order to scan indoor and outdoor environments on the fly, our system has to process every incoming scan in real-time. For our setup, this requires to process every incoming scan within less than 50 ms, corresponding to the maximum scanner frequency of 20 Hz. To evaluate these constraints, we analyzed the latency estimates provided by our hardware development tool and the runtime measurements of our SLAM-Box during the scanning process. The hardware design achieved a clock frequency of 150 MHz for registration and 100 MHz for the TSDF kernel. This results in a total latency of 16.83 ms for the TSDF kernel and a total latency of 9.45 ms for the registration kernel. When analyzing the kernel implementation in more detail, it can be seen that the TSDF projection

Table 6: Runtime measurements for the algorithm during an indoor scanning process for different scan line configurations of the OS1-128.

	Registration			TSDF Update		
lines	mean [s]	min [s]	max [s]	mean [s]	min [s]	max [s]
16	0.038	0.008	0.143	0.184	0.131	0.223
32	0.059	0.010	0.301	0.229	0.116	0.285
64	0.072	0.010	0.504	0.321	0.172	0.419
128	0.104	0.011	0.557	0.380	0.261	0.482

and interpolation phase has a latency of 6.8 ms, while a latency of 10 ms for the map merging part of the TSDF kernel is achieved. Hence, generating a new TSDF volume based on new sensor data takes less time for computation than integrating a new map part into the already existing representation due to more memory access via the DRAM ports. Furthermore, it can be observed that the TSDF kernel takes about twice as much time as the registration kernel. This is why decoupling of both procedures is crucial to ensure the real-time capability of the SLAM-Box.

The runtime of the accelerated algorithm was measured during the experiments in different indoor scenarios for both the Velodyne VLP-16 and the Ouster OS1-128 lidar sensors, respectively. Our system hardware was initially optimized for the first sensor. The corresponding histogram for the measured runtimes using VLP-16 is visualized in Fig. 14. It shows, that only 11 % of the scans are processed slower than the scanner frequency. The average processing time for an individual scan was $30\,ms$, much faster than the maximum sensor frequency. This leads to the conclusion that on average the SLAM-box is able process the incoming sensor data in real time. This can also be observed during scanning, where the SLAM-Box was able to localize itself correctly without delay, resulting in accurate maps. Furthermore we tested the limits of our system by increasing the number of scan lines, received by the sensor. This is done

using the OS1-128 and the results are depicted in Table 6. As can be seen, the runtime of the system continuously increases with the number of considered scan lines. Although it can be noticed that beginning with 32 scan lines, the algorithm is not able to execute the localisation faster than the maximum frequency of the sensor, the system could track its position during the complete scanning process. This is because of the slower movement speed during the experiment, which decreases the change in position that must be determined between two registration steps and also the number of TSDF updates that are needed to keep the map consistent. Furthermore it can be noticed that the maximum runtime for the registration and the TSDF update are significantly higher than the average runtime of both parts of the algorithm. For the registration, this occurs during a change in rotation. In this case, more iterations are needed to determine the correct transformation from the current to the last frame. Regarding the TSDF update, the higher maximum runtime can be explained by considering the projection method of the TSDF calculation. In this step, the intersection of a ray with every cell between the origin of the sensor and all scan points must be calculated to update the map correctly. Therefore, the number of updated cells increases with distance of the scan points to the origin of the scanner. This results in a lower performance of the TSDF update for scenarios with larger free areas.

6.5. Power Consumption

To quantify the energy efficiency of HATSDF SLAM, we used an oscilloscope and a LowPowerLab Current Ranger to measure the current over time to trace the power consumption of individual processes as visualized in Fig. 15 and shown in Tab. 7. The measurement only considers the FPGA board and the SSD. Most power is consumed by the TSDF update step, which occurs approximately between t_1 and t_2. It is important to mention that the map update and the registration process are performed in parallel on the system. This results in a higher total power consumption. The registration only causes small peaks due to the efficient hardware acceleration. This can also be noticed between t_2 and t_3,

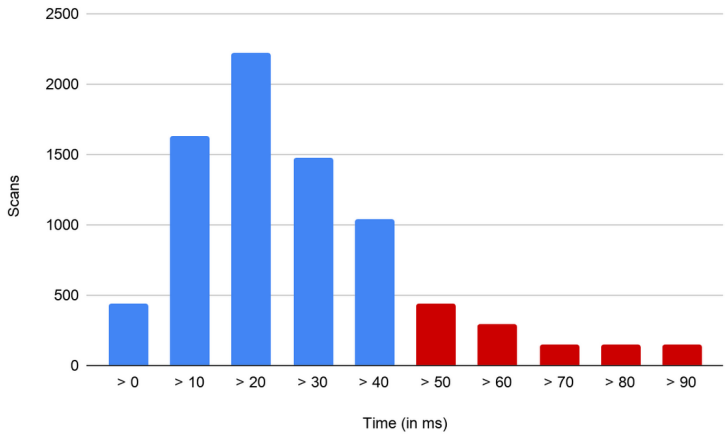


Figure 14: Histogram of scan integration times. The red buckets mark the times, which are slower than the maximum scanner frequency of 20 Hz.

Table 7: Measured current and power of the SLAM-Box in idle and running mode at a supply voltage of 12 V.

	Idle			Running		
	mean	min	max	mean	min	max
I [A]	0.95	0.92	1.07	1.15	1.03	1.27
P [W]	11.43	11.04	12.78	**13.80**	12.36	15.24

where only the registration kernel is executed on the FPGA. The board has an average power consumption of 13.8 W with lower and upper bounds 12.36 W and 15.24 W. For reference, we compared it with an Intel NUC (NUC6i7KYK, Core i7-6770HQ), normally used on our robots, cf. Tab. 8. Our system consumes less energy while being able to process the data in a shorter time. Overall, 18 times less energy is required to process one frame with HATSDF SLAM compared to the reference system.

7. Discussion and Outlook

HATSDF SLAM, a TSDF SLAM implementation targeting mobile robots with limited energy budget has been presented. By efficiently utilizing embed-

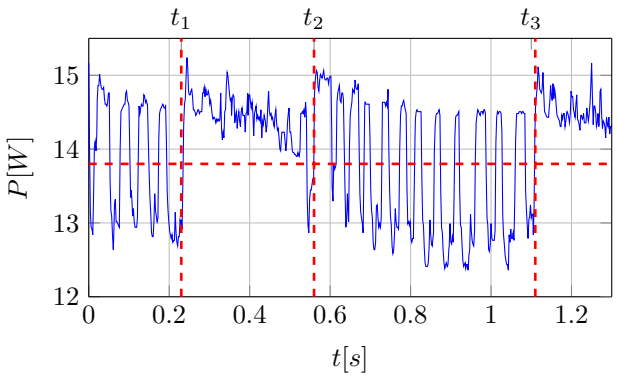


Figure 15: Power consumption of the HATSDF SLAM box over time. The horizontal line at $13.8\,W$ marks the mean over the measured period.

Table 8: Runtime and power consumption in software and hardware.

Platform	Runtime [s]	Power [W]	Energy [J/Scan]
NUC	0.279	34.0	9.49
SLAM-Box	0.038	13.8	0.52

ded ARM processors and FPGA-based hardware accelerators on a reconfigurable SoC, we achieved real-time performance with eighteen times smaller energy requirements per frame compared to a state-of-the-art PC. The integrated ROS bridge enables not only online inspection of the mapping process but also offline evaluation of reference data sets. In all scenarios, HATSDF SLAM compares well to established SLAM algorithms, providing accurate pose estimation with minimal drift. Due to its modular architecture, the implementation can be easily adapted to new sensors or algorithmic modifications. Next steps will include the utilization of hardware reconfiguration at runtime, enabling the system to automatically adapt to changing environmental conditions. The goals of such further development include increasing the resolution of the TSDF volume while reducing the registration time for high resolution LiDAR data. This may become possible by combining the approach of Kintinous with the idea of exploiting free spaces in the environment by implementing a more memory

efficient data structure to represent the local map. Because of the higher complexity of these data structures compared to the current map representation, it is expected that this will lead to a higher resource utilization in terms of logic units on the FPGA. However, since these resources are already highly utilized, further design decisions must be made to implement more efficient environment representations in hardware. Moreover, feature-based SLAM methods may be a promising approach to speed up the registration process.

Acknowledgements

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

References

[1] M. R. U. Saputra, A. Markham, N. Trigoni, Visual SLAM and structure from motion in dynamic environments: A survey, ACM Computing Surveys (CSUR) 51 (2) (2018) 1–36.

[2] C. Debeunne, D. Vivet, A review of visual-LiDAR fusion based simultaneous localization and mapping, Sensors 20 (7) (2020) 2068.

[3] M. Eisoldt, M. Flottmann, J. Gaal, P. Buschermöhle, S. Hinderink, M. Hillmann, A. Nitschmann, P. Hoffmann, T. Wiemann, M. Porrmann, Hatsdf slam–hardware-accelerated tsdf slam for reconfigurable socs, in: 2021 European Conference on Mobile Robots (ECMR), IEEE, 2021, pp. 1–7.

[4] M. Flottmann, M. Eisoldt, J. Gaal, M. Rothmann, M. Tassemeier, T. Wiemann, M. Porrmann, Energy-efficient fpga-accelerated lidar-based slam for embedded robotics, in: 2021 International Conference on Field-Programmable Technology (ICFPT), IEEE, 2021, pp. 1–6.

[5] S. Pütz, T. Wiemann, J. Sprickerhof, J. Hertzberg, 3d Navigation Mesh Generation for Path Planning in Uneven Terrain, IFAC-PapersOnLine 49 (15) (2016) 212–217. doi:10.1016/j.ifacol.2016.07.734.

[6] T. Wiemann, K. Lingemann, J. Hertzberg, Automatic Map Creation for Environment Modelling in Robotic Simulators, in: Proceedings of the European Conference on Modelling and Simulation (ECMS), 2013.

[7] P. J. Besl, N. D. McKay, Method for registration of 3-d shapes, in: Sensor fusion IV: control paradigms and data structures, Vol. 1611, International Society for Optics and Photonics, 1992, pp. 586–606.

[8] A. Segal, D. Haehnel, S. Thrun, Generalized-icp., in: Robotics: science and systems, Vol. 2, 2009, p. 435.

[9] K.-L. Low, Linear least-squares optimization for point-to-plane icp surface registration, Chapel Hill, University of North Carolina 4 (10) (2004) 1–3.

[10] D. Holz, S. Behnke, Registration of non-uniform density 3d point clouds using approximate surface reconstruction, in: ISR/Robotik 2014; 41st International Symposium on Robotics, VDE, 2014, pp. 1–7.

[11] J. Zhang, S. Singh, Loam: Lidar odometry and mapping in real-time., in: Robotics: Science and Systems, Vol. 2, 2014.

[12] T. Shan, B. Englot, LeGO-LOAM: Lightweight and Ground-Optimized Lidar Odometry and Mapping on Variable Terrain, in: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018, pp. 4758–4765. doi:10.1109/IROS.2018.8594299.

[13] W. E. Lorensen, H. E. Cline, Marching Cubes: A High Resolution 3d Surface Construction Algorithm, in: ACM SIGGRAPH '87, 1987.

[14] S. Izadi, R. A. Newcombe, D. Kim, O. Hilliges, D. Molyneaux, S. Hodges, P. Kohli, J. Shotton, A. J. Davison, A. Fitzgibbon, KinectFusion: Real-time Dynamic 3d Surface Reconstruction and Interaction, in: ACM SIGGRAPH 2011 Talks, ACM, 2011.

[15] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, J. McDonald, Kintinuous: Spatially extended KinectFusion, in: RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras, Sydney, Australia, 2012.

[16] M. Nießner, M. Zollhöfer, S. Izadi, M. Stamminger, Real-time 3d reconstruction at scale using voxel hashing, ACM Transactions on Graphics (ToG) 32 (6) (2013) 1–11.

[17] Q. Gautier, A. Shearer, J. Matai, D. Richmond, P. Meng, R. Kastner, Real-time 3D Reconstruction for FPGAs: A Case Study for Evaluating the Performance, Area, and Programmability Trade-offs of the Altera OpenCL SDK, in: FPT Conference, 2014, pp. 326–329. doi:10.13140/RG.2.1.4950.4168.

[18] A. Kosuge, K. Yamamoto, Y. Akamine, T. Oshima, An SoC-FPGA-Based Iterative-Closest-Point Accelerator Enabling Faster Picking Robots, IEEE Transactions on Industrial Electronics 68 (4) (2021) 3567–3576. doi:10.1109/TIE.2020.2978722.

[19] Q. Gautier, A. Althoff, R. Kastner, FPGA Architectures for Real-time Dense SLAM, 2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP) 2160-052X (2019) 83–90.

[20] D. Törtei, J. Piat, M. Devy, FPGA design and implementation of a matrix multiplier based accelerator for 3D EKF SLAM, 2014 International Conference on Reconfigurable Computing and FPGAs (ReConFig) (2014). doi:10.1109/ReConFig.2014.7032523.

[21] K. Boikos, C. Bouganis, Semi-dense SLAM on an FPGA SoC, 2016 26th International Conference on Field Programmable Logic and Applications (FPL) (2016) 1–4.

[22] K. Boikos, C. Bouganis, A high-performance system-on-chip architecture for direct tracking for SLAM, 2017 27th International Conference on Field Programmable Logic and Applications (FPL) (2017) 1–7.

[23] K. Boikos, C. Bouganis, A Scalable FPGA-based Architecture for Depth Estimation in SLAM, in: ARC, 2019.

[24] D. R. Canelhas, T. Stoyanov, A. J. Lilienthal, SDF tracker: A parallel algorithm for on-line pose estimation and scene reconstruction from depth images, in: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2013, pp. 3671–3676.

[25] T. Wiemann, F. Igelbrink, S. Pütz, J. Hertzberg, A file structure and reference data set for high resolution hyperspectral 3d point clouds, IFAC-PapersOnLine 52 (8) (2019) 403–408.

[26] T. Wiemann, I. Mitschke, A. Mock, J. Hertzberg, Surface reconstruction from arbitrarily large point clouds, in: 2018 Second IEEE International Conference on Robotic Computing (IRC), 2018, pp. 278–281. doi:10.1109/IRC.2018.00059.

[27] O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl, H.-P. Seidel, Laplacian surface editing, in: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing, 2004, pp. 175–184.