HATSDF SLAM – Hardware-accelerated TSDF SLAM for Reconfigurable SoCs

Marc Eisoldt*, Marcel Flottmann*, Julian Gaal*, Pascal Buschermöhle*, Steffen Hinderink*, Malte Hillmann*, Adrian Nitschmann*, Patrick Hoffmann*, Thomas Wiemann*†, Mario Porrmann*

*Osnabrück University, Institute of Computer Science, Osnabrück, Germany
Email: firstname.lastname@uni-osnabrueck.de
†DFKI Niedersachsen Lab, Plan Based Robot Control, Osnabrück, Germany
Email: thomas.wiemann@dfki.de

***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 a Velodyne VLP-16 LiDAR 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.**

I. 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 to track 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.

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

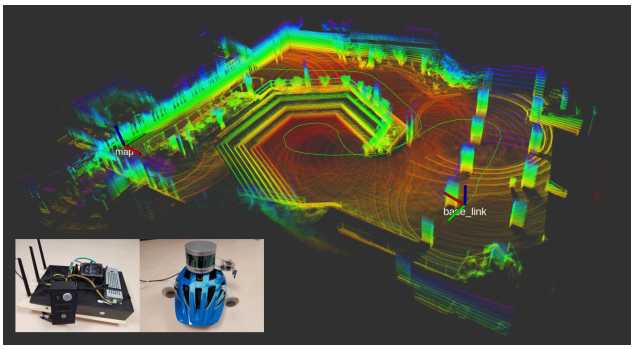

Fig. 1. The HATSDF hardware consisting of a head-mounted Velodyne VLP-16 LiDAR and mobile embedded computing system (reconfigurable SoC). The TSDF-based registration to generate the map of an urban environment was done online and in real time using the SoC's CPUs and FPGA. The overall power consumption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

In this paper, we present an approach called HATSDF (Hardware Accelerated TSDF SLAM) for incremental SLAM using 3D depth sensors that is able to integrate the incoming data of a Velodyne VLP-16 laser scanner in real time to maintain a 3D environment map[1]. The 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 [3] or as environment representation in simulators [4]. The proposed algorithm is designed to run on reconfigurable Systems-on-Chips (SoCs) that 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 system presented in this paper is realized as a stand-

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

alone SLAM box with an optional ROS interface that fuses data from the Velodyne VLP-16 LiDAR and an integrated IMU in real time. Additionally, pre-recorded sensor data from ROS bag files can be fed into the system for parameter tuning and benchmarking. We show that our system is able to integrate the LiDAR data in real time. 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.

II. Related Work

Incremental SLAM with 3D point clouds is usually solved with some version of the well-known ICP algorithm [5], that computes the transformation matrix between sets of corresponding points in two point clouds. An often-used extension is the Generalized ICP algorithm [6], 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 [7] or Point-Mesh-Correspondences [8]. 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) [9] have been developed for that special purpose. The well-known LeGO-LOAM approach [10] 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 [11]. In context of SLAM, KinectFusion [12] 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 [13] extended that to larger environments using a clever swapping strategy. However, these algorithms are tailored for RGB-D or time-of-flight cameras, which usually have a limited range and are sensitive to ambient light. Also, 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 these accelerators in the context of SLAM. In [14], FPGAs where used in the context of EKF-SLAM allowing to track up to 48 landmarks in real time. A method for ICP acceleration with graph-based nearest neighbor search on an FPGA was shown in [15]. However, the system was only able to achieve 1 FPS on scans with 30.000 points. A method for KinectFusion-like algorithms on FPGAs was shown in [16]. It achieved a performance of about 3 frames per second.

These methods are prone to drift without external reference (e.g., GPS). Usually this issue is overcome by offline optimization of the pose estimates with graph-based approaches and loop closing [17, 18, 19]. Our main contribution is an incremental FPGA-based TSDF SLAM approach implemented on an integrated System on Chip (SoC) to create large-scale TSDF maps from LiDARs. It is neither dependent on odometry nor GPS and shows only low drift due to two factors: Firstly, it is able to process point clouds of a Velodyne VLP-16 LiDAR in real time at a rate exceeding the sensor frame rate of 20 Hz. Secondly, we use a Point-to-TSDF registration method [20] that is known to be precise, but computationally expensive on CPUs. To address this problem, we implemented a hardware-accelerated version of this approach. 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 an integrated standalone sensor system ("SLAM box") while walking around and offline comparison with publicly available data sets. The 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.

III. 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 [13] to an FPGA accelerated version. Individual map parts, so called local maps, are calculated from a set of several scan frames. If the spatial region of a local map exceeds a predefined maximum, it is fused into a global map representation using a weighted average after successful registration. To register

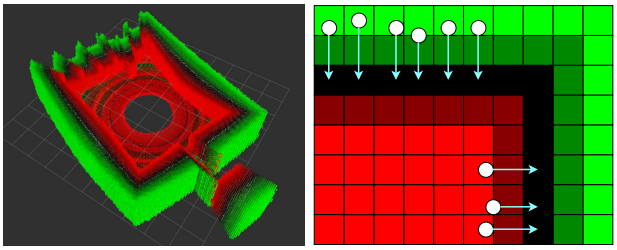

Fig. 2. Example of a TSDF voxel grid derived from a single laser scan (left) and an illustration of the Point-to-TSDF approach used in this paper (right). Green voxels represent positive TSDF values, red voxels negative ones. The turquoise arrows visualize the transformation, which is aspired by the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

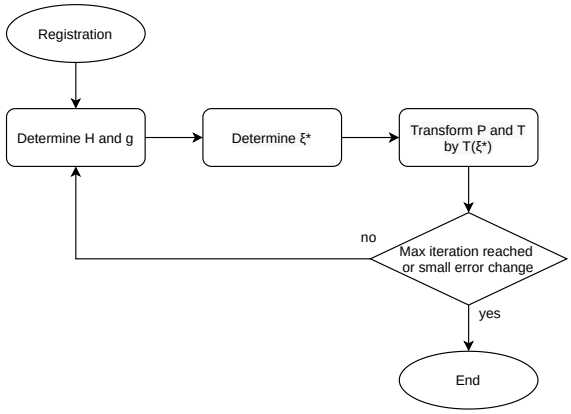

Fig. 3. Flow diagram of the iterative Point-to-TSDF registration procedure

the incoming frames with the local map, we use a Point-to-TSDF approach [20] 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 system architecture and actual implementation.

IV. System Architecture

A. Design Goals

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 robotics 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

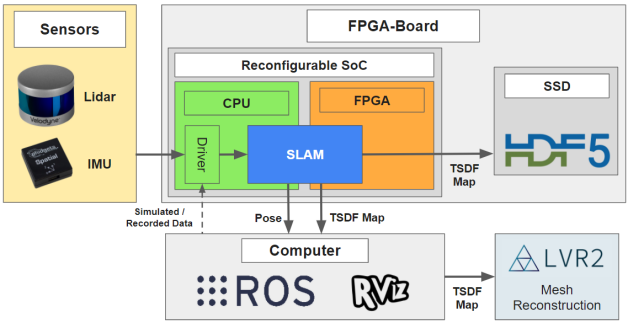

Fig. 4. System architecture of the SLAM-Box.

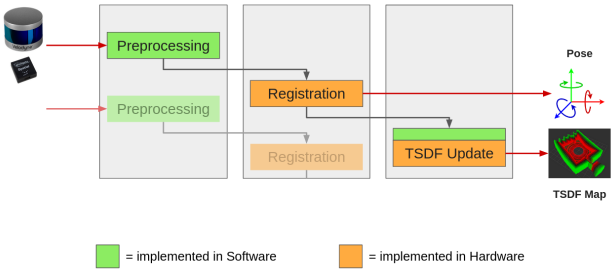

Fig. 5. Pipelining of the HATSDF SLAM. Preprocessing, registration and TSDF map update are pipelined to maximize data throughput.

evaluation, as our system is placed in a backpack and the sensors are mounted on a helmet.

Fig. 4 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 [21]. The following subsections describe the system architecture in detail.

B. FPGA Board

We used a board from Trenz Electronics, based on a Xilinx Zynq UltraScale+ ZU15EG SoC, integrating Arm CPU cores and an FPGA fabric. It provides the basic communication infrastructure like Ethernet (LiDAR and external communication), USB (IMU), and SATA (SSD). 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.

C. 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 within the ROS ecosystem. 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 own 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.

D. 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 [21] 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 algorithm can be used to compute a triangle mesh, which is post-processed with Laplacian smoothing and a hole filling algorithm to remove artifacts.

E. Pipelining

HATSDF SLAM mainly consists of three processing steps, namely 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 amount of points in the point cloud while simultaneously keeping enough information to obtain good results in the registration step. The last task of the pre-processing step is to accumulate the IMU data published between the last scan and the current scan to get an initial rotation estimation. The

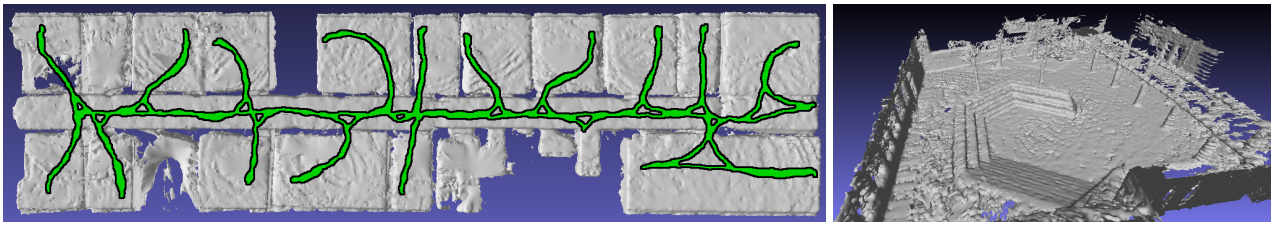

Fig. 6. Online evaluation of the HATSDF SLAM in a building interior (left) and an urban outdoor location (right). The images show the reconstructed mesh from the generated TSDF representation. The trajectory in the building data is rendered in green. The indoor data set has a dimension of approximately 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 $80\,m$.

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. 5. For this, each processing step is executed in its own thread. This allows new scans to be processed right away instead of having to wait for the previous scan to finish. It is worth noting that the TSDF map update is not actually calculated for every scan, because it is by far the most time-consuming step. Reducing the amount of TSDF map updates is justifiable considering that our system processes scans at a high frequency which leads to only small information gains of the scanner's surrounding per scan.

V. Evaluation

To evaluate our approach, we used our system online in different environments (indoors and outdoors), as shown in Fig. 6. Furthermore, we used the ROS bridge to feed prerecorded data into our system to assess the performance on available reference data sets with known ground-truth. In order to allow reproducibility, the descriptions of all system parameters and the configuration files for every considered dataset are provided together with the source code.

A. Online SLAM in Real Time

To test the system with structured data, we scanned an entire floor of an office building (about $500\,m^2$). This structured environment features planar walls, floor and ceilings. The head-mounted LiDAR was taken along the green path shown left in Fig. 6, visiting every room before returning to the starting position. The total length of the recorded path is about $275\,m$. Scan time was 6:52 minutes and 8098 scans were inserted into the TSDF map, which was then reconstructed into a mesh. To measure the accuracy of the registration and 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 $7.5\,cm$, the rotation varied by less than 2^o in Euler angles, which is within the order of magnitude of one voxel ($6.4\,cm$) used for all online datasets. Mapping was done without any external pose estimates.

As SLAM is considered to work better in structured environments with plane surfaces, we further evaluated our approach in a less structured outdoor environment. These environments are generally more challenging because of the lack of floors and ceilings to register with. As our scanner's vertical field of view is only 30 degrees wide, we only see the ground about $7\,m$ away from the sensor, depending on the height of the person wearing the backpack and helmet. We therefore needed to reduce the scanner's distance to the ground in order for it to see enough of the ground to register with. This was achieved by driving a kart instead of walking. Another challenge our approach faced in the outdoor environment were slopes that changed the height of the laser scanner. In total, we recorded 3494 scans in 2:57 minutes and covered approximately $80\,m$. The resulting mesh map is shown in the right part of Fig. 6. It demonstrates that our approach is able to master height changes, as they are clearly visible in the resulting map.

B. 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. LOAM has a similar measurement setup using a VLP-16 scanning running at 10 Hz and an 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 LOAM and the HATSDF SLAM were plotted against each other. The result is shown in the left part of Fig. 7, with the HATSDF SLAM path rotated by 6 degrees to compensate an initial rotational offset. 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 faster compared to our scenarios. Also, no IMU data is provided in the odometry data sets. In contrast to LOAM, ground truth for the pose path is available. Due to more sensor data per scan frame and larger distances, in this scenario the ICP step in HATSDF SLAM needs more

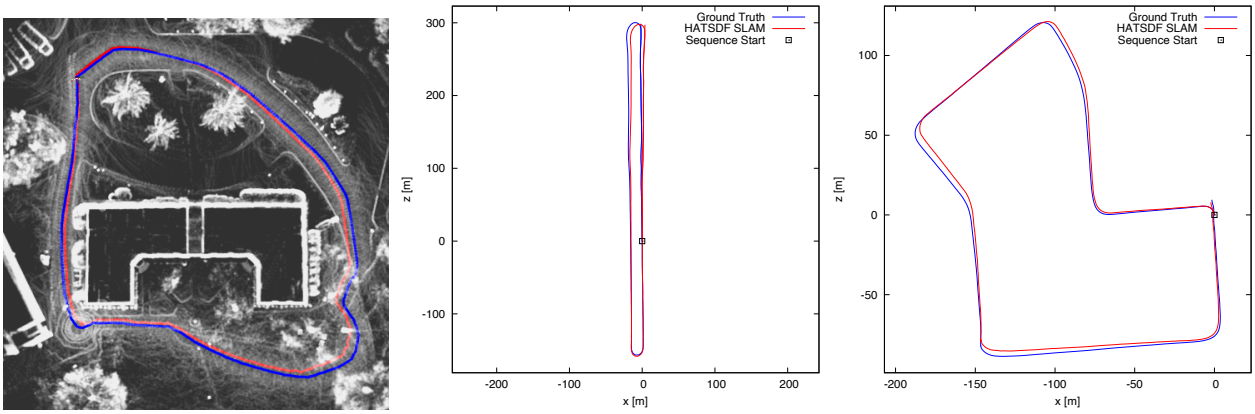

Fig. 7. Offline comparison of HATSDF SLAM with reference data sets and trajectories from LeGO-LOAM SLAM [10] (left) and the KITTI data sets 6 (middle) and 7 (right). Pose paths estimated by the HATSDF SLAM are shown in red, reference trajectories are displayed in blue.

iterations to converge. To compensate this, we reduced the data rate by a factor of 10 as compared to the original data, giving our algorithm enough time to converge. The estimated trajectories for data set 6 and 7 are visualized in the middle and the right side of Fig. 7. Though 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. Given that limitation, our results reflect the driven path well.

C. Runtime

The runtime of the SLAM-Box was measured during the online indoor test described in Sec. V-A and is visualized in Fig. 8. It shows, that only $11\,\%$ of the scans are processed slower than the scanner frequency of $20Hz$. 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 a delay, resulting in accurate maps.

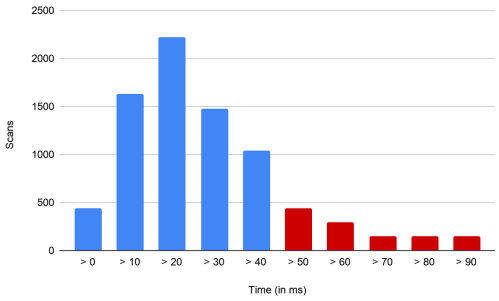

Fig. 8. Histogram of scan integration times. The red buckets mark the times, which are slower than the maximum scanner frequency of 20 Hz

D. 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. 9 and shown in Tab. I. Most power is consumed by the TSDF update step. The registration only causes a small peak due to the efficient hardware acceleration. 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. II. 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.

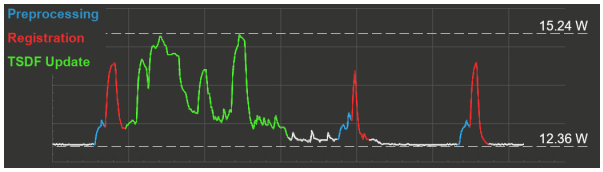

Fig. 9. Power consumption of sub-routines of the hardware implementation.

TABLE I
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

VI. DISCUSSION AND OUTLOOK

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

TABLE II
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

and FPGA-based hardware accelerators on a reconfigurable SoC, we achieved real-time performance with eighteen times less energy 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.

REFERENCES

[1] M. R. U. Saputra, A. Markham, and N. Trigoni, "Visual SLAM and structure from motion in dynamic environments: A survey," *ACM Computing Surveys (CSUR)*, vol. 51, no. 2, pp. 1–36, 2018.

[2] C. Debeunne and D. Vivet, "A review of visual-LiDAR fusion based simultaneous localization and mapping," *Sensors*, vol. 20, no. 7, p. 2068, 2020.

[3] S. Pütz, T. Wiemann, J. Sprickerhof, and J. Hertzberg, "3d Navigation Mesh Generation for Path Planning in Uneven Terrain," *IFAC-PapersOnLine*, vol. 49, no. 15, pp. 212–217, 2016.

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

[5] P. J. Besl and 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.

[6] A. Segal, D. Haehnel, and S. Thrun, "Generalized-icp." in *Robotics: science and systems*, vol. 2, no. 4, 2009, p. 435.

[7] K.-L. Low, "Linear least-squares optimization for point-to-plane icp surface registration," *Chapel Hill, University of North Carolina*, vol. 4, no. 10, pp. 1–3, 2004.

[8] D. Holz and 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.

[9] J. Zhang and S. Singh, "Loam: Lidar odometry and mapping in real-time." in *Robotics: Science and Systems*, vol. 2, no. 9, 2014.

[10] T. Shan and 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)*, Oct. 2018, pp. 4758–4765.

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

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

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

[14] D. T. Tertei, J. Piat, and M. Devy, "FPGA design of ekf block accelerator for 3d visual SLAM," *Computers & Electrical Engineering*, vol. 55, pp. 123–137, 2016.

[15] A. Kosuge, K. Yamamoto, Y. Akamine, and T. Oshima, "An SoC-FPGA-Based Iterative-Closest-Point Accelerator Enabling Faster Picking Robots," *IEEE Transactions on Industrial Electronics*, vol. 68, no. 4, pp. 3567–3576, 2020.

[16] Q. Gautier, A. Althoff, and R. Kastner, "FPGA architectures for real-time dense SLAM," in *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, vol. 2160. IEEE, 2019, pp. 83–90.

[17] D. Borrmann, J. Elseberg, K. Lingemann, A. Nüchter, and J. Hertzberg, "Globally consistent 3d mapping with scan matching," *Robotics and Autonomous Systems*, vol. 56, no. 2, pp. 130–142, 2008.

[18] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based SLAM," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31–43, 2010.

[19] X. Chen, T. Läbe, A. Milioto, T. Röhling, O. Vysotska, A. Haag, J. Behley, C. Stachniss, and F. Fraunhofer, "OverlapNet: Loop closing for LiDAR-based SLAM," in *Proc. of Robotics: Science and Systems (RSS)*, 2020.

[20] D. R. Canelhas, T. Stoyanov, and 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.

[21] T. Wiemann, I. Mitschke, A. Mock, and J. Hertzberg, "Surface reconstruction from arbitrarily large point clouds," in *2018 Second IEEE International Conference on Robotic Computing (IRC)*, Jan 2018, pp. 278–281.