Energy-efficient FPGA-accelerated LiDAR-based SLAM for Embedded Robotics

Marcel Flottmann[1], Marc Eisoldt[1], Julian Gaal[1], Marc Rothmann[1], Marco Tassemeier[1], Thomas Wiemann[1,2], Mario Porrmann[1]

***Abstract*—Being one of the fundamental problems in autonomous robotics, SLAM (Simultaneous Localization and Mapping) algorithms have gained a lot of attention. Although numerous approaches have been presented for determining 6D poses in 3D environments, one of the main challenges that remains is the required combination of real-time processing and high energy efficiency. In this paper, a combination of CPU and FPGA processing is used to tackle this problem, utilizing a reconfigurable SoC. We present a complete solution for embedded LiDAR-based SLAM that uses a global Truncated Signed Distance Function (TSDF) as map representation. A hardware-in-the-loop environment with ROS integration enables efficient evaluation of new variants of algorithms and implementations. Based on benchmark data sets and real-world environments, we show that our approach compares well to established SLAM algorithms. Compared to a software implementation on a state-of-the-art PC, the proposed implementation achieves a 7-fold speed-up and requires 18 times less energy when using a Xilinx UltraScale+ XCZU15EG.**

***Index Terms*—Embedded LiDAR-based SLAM, FPGA acceleration, TSDF map, autonomous robotics**

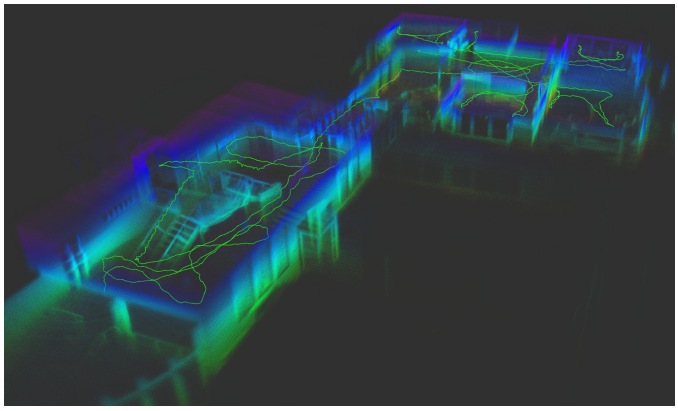

Fig. 1: Visualization of the sensor data recorded in an indoor environment in real-time using a Xilinx Zynq UltraScale+

I. Introduction

Simultaneous Localization and Mapping (SLAM) in 3D is one of the fundamental problems in autonomous robotics. The main goal is to create a 3D map of an environment previously unknown to the robot. SLAM is a necessary prerequisite for other tasks in this domain, such as navigation and action planning. Since the map is constructed by several scans captured from different positions, the robot must be able to track its own 6D pose (position and orientation in 3D) during the mapping procedure. Several solutions have been established based on different scanner systems, ranging from monocular SLAM using RGB cameras [1] to the use of 3D information from LiDAR (Light Detection And Ranging) or similar sensors [2]. The main challenge of SLAM is that miss-alignments of the scan data lead to a continuous deterioration of the pose estimation over time and thus, a consistent integration of all sensor information during data acquisition is of vital importance.

Besides real-time capability, energy efficiency is a critical aspect for battery-powered autonomous robots. Most SLAM algorithms are originally developed for powerful desktop CPUs and GPUs. Popular examples for indoor applications with RGB-D cameras are the KinectFusion [3] algorithm and the extension Kintinuous [4], which have been implemented on GPUs and meet real-time requirements. However, utilizing GPUs significantly increases the power consumption. For autonomous robots with limited energy budget, new approaches are required utilizing, e.g., embedded GPUs or FPGAs. In recent years, there have been several publications addressing the implementation of SLAM algorithms for mobile systems utilizing embedded processors in combination with hardware accelerators, often implemented in FPGAs [5]–[10].

In this paper, we introduce a novel FPGA-based hardware architecture to accelerate an online incremental SLAM approach in order to achieve a 3D map in real-time. For this purpose, a Velodyne VLP-16 laser scanner is used to generate a Truncated Signed Distance Function (TSDF) volume as map representation. Based on the TSDF map, a polygonal mesh can be efficiently generated for use in other domains such as path planning [11] or environment simulations [12]. The complete hardware setup, including laser scanner, an Inertial Measurement Unit (IMU) for improved rotation estimation, FPGA system, wireless interconnect, and battery power, has been integrated into a compact SLAM-Box, enabling real-time indoor and outdoor scans. Fig. 1 shows an example map that has been generated with the system.

[1]The authors are with the Computer Engineering and Autonomous Robotics groups at the Institute of Computer Science, Osnabrück University, Osnabrück, Germany firstname.lastname@uni-osnabrueck.de

[2]Thomas Wiemann is with the German Research Center for Artificial Intelligence (DFKI), Niedersachsen Lab, Plan-based Robot Control Group, Osnabrück, Germany thomas.wiemann@dfki.de

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

II. Application Scenario

In our scenario, efficient access to the map is crucial to ensure a fast localization and map update. Since these maps are very large (typically GBytes), intelligent memory management is required. For this purpose, the originally GPU-optimized swapping strategy of Kintinuous [4] is implemented on our FPGA accelerated architecture. The main idea is to distinguish between a local map and a global map. The global map is stored persistently on the disc in a Hierarchical Data Format (HDF5). In addition to that, the local map stores a small, previously loaded part of the complete map, representing a small area around the current position of the system with faster access to the required data. Hence, if the system changes its position during the mapping process frequently, the local map can be shifted very efficiently.

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 in [13]. 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 orthogonally 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 [13] to determine the transformation between successive poses, where a newly acquired scan is registered using the most recent local map, as detailed in [14]. This task can be described as a minimization problem and be solved iteratively with the help of an initial rotation estimation provided by the IMU. The procedure can be derived from the defined error function using a first-order Taylor series approximation. Thus, the correctness of the solution is only given for transformations close to the unit matrix. Hence, the closer the movement between two time steps, the better this condition is fulfilled, and the better is the accuracy of the pose. Both the mapping and the localization procedure show high potential for parallelization and can therefore be efficiently accelerated on reconfigurable hardware.

III. System Architecture and Design Flow

Often, SLAM algorithms either focus on registration only or use power-hungry PCs or GPUs for live mapping. Mobile robots have significant power constraints and therefore require an efficient system architecture to process live data. Therefore, our goal was to develop a standalone compute system for power efficient and accurate localization and mapping, independent of specific LiDAR sensors. The architecture of the resulting SLAM-Box is shown in Fig. 2.

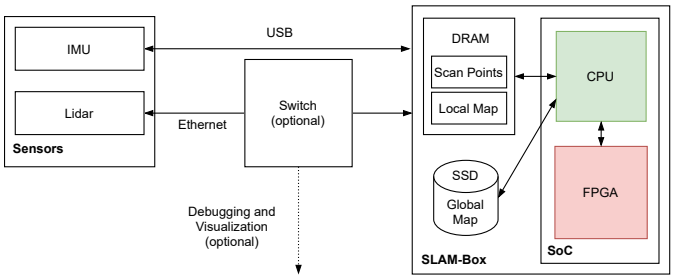

Fig. 2: System architecture of the SLAM-Box

A Trenz UltraSOM+ module equipped with a Xilinx XCZU15EG SoC, embedded in an UltraITX+ base board, has been selected as the core of our platform. The combination of a quad Arm Cortex-A53 with a quite large FPGA fabric on the reconfigurable SoC creates a wide range of opportunities for possible hardware-software partitions to accelerate the application. A SATA-based SSD is used for storing the global map, and 4 GByte DDR4 memory is available for the local map. The LiDAR sensor (Velodyne Puck VLP-16) and the IMU (PhidgetSpatial Precision 3/3/3 High Resolution) are connected to the SLAM-Box via Ethernet and USB, respectively.

To demonstrate the flexibility of our approach, in our experimental setup, the Velodyne Puck and IMU were mounted on a helmet that was worn by a team members during the process of mapping an environment (cf. Fig. 3). The SLAM-Box and sensors both have quite similar power requirements

Fig. 3: Physical implementation of the SLAM-Box (FPGA module marked in red) including a WiFi router for debugging and the head-mounted Velodyne VLP-16 LiDAR

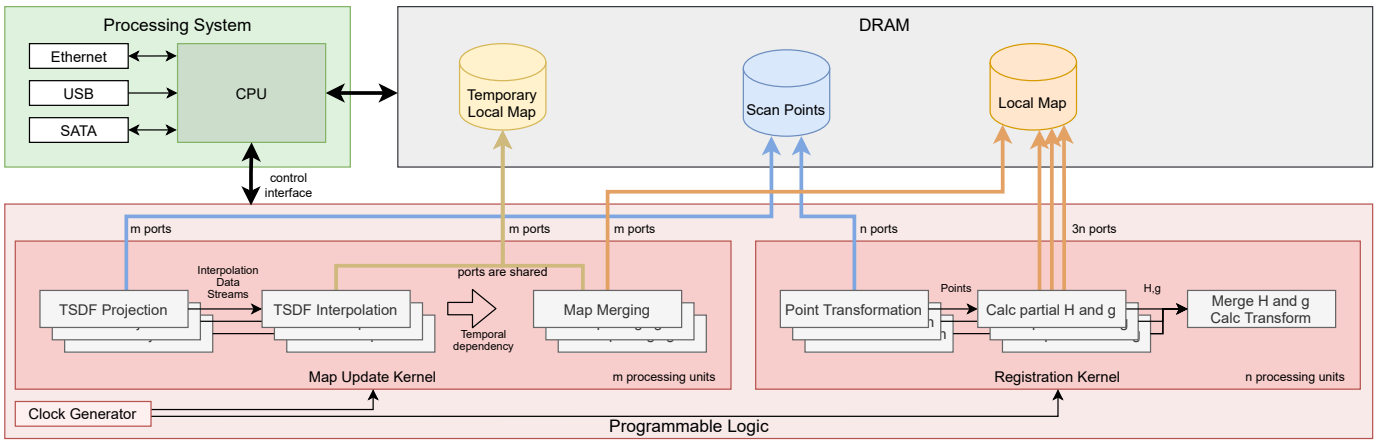

Fig. 4: High-level architecture of the SLAM implementation

and enable localization and mapping tasks spanning several hours. Given these power efficiency characteristics and the SLAM-Box being a standalone device, it is conceivable to mount our system onto drones or other vehicles as well.

For early evaluation, an optimized, multithreaded software implementation has been realizes as a reference design on an Intel NUC (NUC6i7KYK, Core i7-6770HQ). All implementations have been evaluated with simulated as well as real sensor data, using the Robot Operating System (ROS) and its visualization tools [15]. For the hardware/software codesign, the Xilinx Vitis tools have been used. The kernels that have been 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 a PetaLinux from which the kernels are called through OpenCL and fed with their respective required data.

An optional ROS visualization and debugging interface has been embedded into the platform, which does not add any overhead if the system is not in debug mode. For this purpose, a ROS-independent sensor processing software system has been implemented on the reconfigurable SoC, which is able to handle incoming sensor data from custom drivers. For hardware-in-the-loop testing and access to ROS for debugging and visualizing the FPGA processing in the implementation phase, a thin, non-blocking wrapper has been implemented around the ZeroMQ messaging library that is able to send and receive sensor data. Debugging information and sensor data or the estimated pose can be sent from the reconfigurable SoC to the host system, where it is converted into a ROS-compatible format for debugging and visualization. In the same vein, simulated data from Gazebo or data from pre-recorded files can be converted into a format compatible with our platform and sent to the SoC via Ethernet or WiFi. If this bridge to a ROS host is not used, all sensor and mapping data is saved on the onboard SSD exclusively. Being able to use simulated or pre-recorded data is an important feature since it enables validation of the implementation by comparing it to the software prototype, using the same input data.

IV. FPGA Implementation

This section describes the implementation of the SLAM algorithm on the reconfigurable SoC, as depicted in Fig. 4.

A. Algorithm Overview

To achieve maximum concurrency and to increase the throughput of the sensor processing, the basic algorithm is partitioned into different stages. Before the main SLAM procedure can take place, a preprocessing with a median and reduction filter of the sensor data is required to ensure the filtering of noise and outliers. The main algorithm is divided into the mapping and the localization stage. Mapping comprises the steps of shifting the local map to the correct position based on the current pose, calculating the TSDF grid based on the incoming scan data, and the update of the local TSDF volume based on the newly calculated entries. Localization describes the registration of the current scan to the local map.

As discussed in Section II, all three steps are executed in sequence. To ensure that all scans can be processed when they are received, the algorithm is pipelined as visualized in Fig. 5. The overview of the architecture in Fig. 4 shows the kernels in the FPGA with their internal structure and their connections to the DRAM memory. The latter also shows which content of the DRAM is accessed by the ports. Furthermore, the Processing System with the CPU and the interfaces to the periphery and the clock regions on the FPGA are illustrated.

The implementation of the algorithm is memory bound. The main bottleneck is the access to the local map, located in the DDR4 memory. Even the area of the local map, which is only a fraction of the global map, 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. Additionally, the memory accesses are randomly distributed and not sequential, limiting the DRAM memory throughput. Various caching strategies targeting the utilization of embedded memory have been evaluated but showed minimal impact and are therefore currently not implemented.

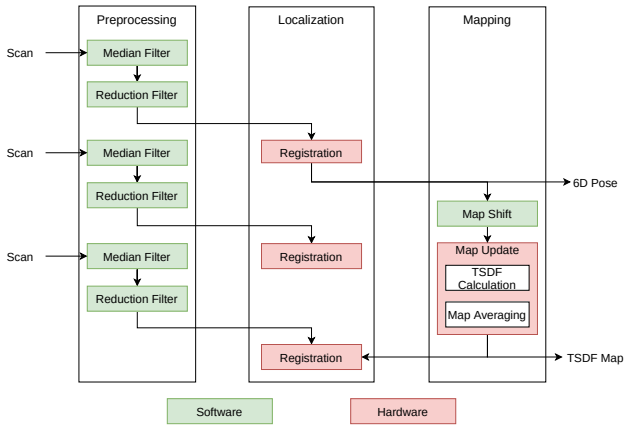

Fig. 5: Visualization of the software pipelining. The algorithm is split up into three threads, each consisting of steps implemented in software or hardware.

B. Execution order of the map update

It could be observed that the map update, which takes much longer than the registration, doesn't have to execute every time, assuming the pose only changes slightly between scans. Hence, a map update is only triggered when one of the following two conditions is met. First, the map update must take place after a predefined number of scans. This ensures that the map is still up to date and reduces the influence of rare outliers. Depending on the average moving speed of the system, the number ranges between 1 for high speed like a running person and 15 for low speed like a slowly moving robot inside a building. Second, after a significant change of position, the map must be updated to ensure correct registration to the map 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 is again configurable and depends on the environment. When the environment contains only sparse features, the threshold should be low, because some features can disappear in the scan while others may reappear. This is why the local map must shift more frequently to ensure that the features are represented in the local map for a robust localization. Using this approach, the system is able to process the incoming data with the maximum scanner frequency of 20 Hz in real-time.

C. Registration Kernel

As shown in Fig. 4, the registration kernel consists of three steps, which are repeated until the error is minimal. The first two steps are the computation-intensive calculations of the H matrix and g vector (cf. [13]). These sub-problems are embarrassingly parallel, and therefore multiple processing units are instantiated for this task. Each unit processes an equally-sized subset of the point cloud. The loop that iterates through the points is pipelined, such that ideally, in every clock cycle, a new point could be processed. The bottleneck in our system is the limited number of six ports to access the DRAM. For each point, seven memory accesses to the local map are needed. Our design uses three units with three memory ports to the local map plus one to the scan points, but the number can be easily adjusted for different FPGAs. This leads to the fact that a new point can be processed only every three clock cycles. Instantiating more than three units does not improve performance since the bandwidth of the ports is fully utilized.

Subsequently, the partial sums from the H and g calculations are merged and the intermediate transformation is calculated. These steps are sequentially dependent. To reduce the latency, all loops in this section are unrolled so that operations that are independent of each other are calculated in parallel. This increases the throughput but also the required FPGA resources.

D. TSDF Update kernel

The TSDF update mainly consists of two phases. First, the new distance values and weights must be calculated and interpolated based on the newly incoming sensor data. Finally, the results must be integrated into the current map.

The first phase is again divided into two steps. Both run in parallel in a pipeline. First, the ray marching is performed in the TSDF projection step for each point in the scan. 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. This data is then sent via a stream to the second step, where the TSDF values are calculated and inserted into a temporary local map. The second phase merges the temporary local map with the actual local map.

Each of the three processing units (projection, interpolation, and merging) can be instantiated multiple times on the FPGA, and therefore the kernel is scalable for different FPGA sizes and memory configurations. For the first phase, the point cloud is split between the corresponding units. For the second phase, the local map is split into equal parts to parallelize the merging. Each unit needs one memory port so that the TSDF update kernel needs a multiple of three of them. On our platform, we instantiated four units each.

V. Evaluation

In this section, the FPGA utilization, performance, and power consumption of the SLAM-Box are evaluated.[1] For this purpose, we use both the information provided by Vivado and time and power measurements when using the SLAM-Box for complete scan procedures in indoor and outdoor environments. Additionally, the hardware implementation is compared to the multithreaded reference design on an Intel NUC (NUC6i7KYK, Core i7-6770HQ) normally used on our robots. The datasets were prerecorded and used as input data for both systems to ensure comparability and reproducibility.

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

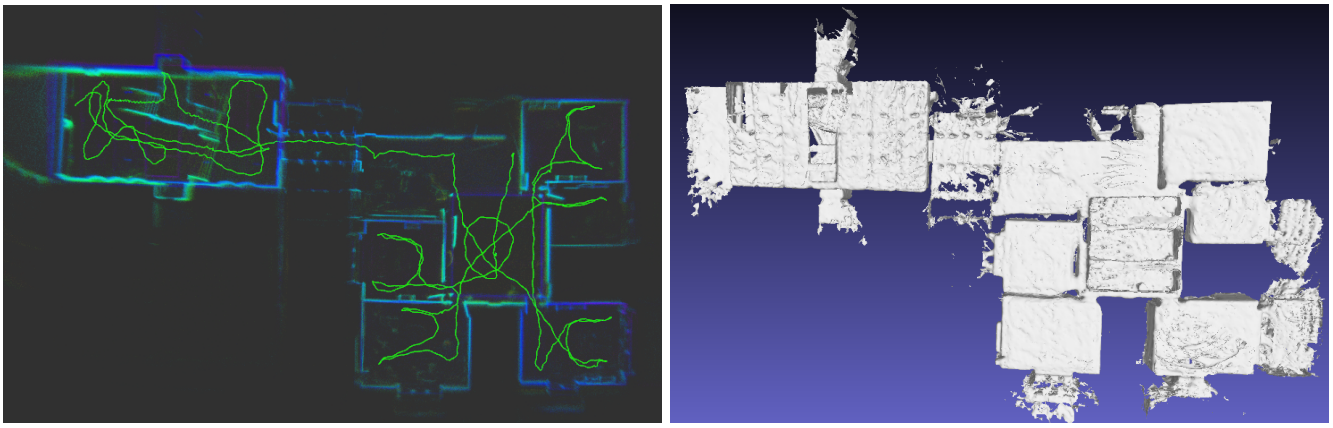

Fig. 6: Top view of the sensor data recorded during the complete scan process (left) and the resulting 3D surface generated based on the TSDF volume. The complete walked trajectory is marked as light green path.

A. Resource Requirements and Performance

The resource utilization of the FPGA design is summarized in Table I. Registration and TSDF Kernel contribute most to the total resources. The rest is required for the base design, including, e.g., the communication infrastructure.

TABLE I: Resource utilization of the Xilinx UltraScale+ XCZU15EG FPGA

	Registration Kernel	TSDF Kernel	Total
CLB [%]	35.87	46.30	98.95
LUT (Logic) [%]	23.52	29.63	61.70
LUT (MEM) [%]	1.71	4.88	15.29
BRAM [%]	4.30	2.69	6.99
DSPs [%]	18.37	18.88	37.25

An important feature of our SLAM-Box is the ability to process every incoming scan in real-time. In the current setup, this means that the implementation must process a complete scan within 50 ms, corresponding to the maximum scanner frequency of 20 Hz. To analyze the performance of our system, we consider both the latency estimations provided by Vivado and the runtime measurements of our SLAM-Box. A clock frequency of 150 MHz for the registration and 100 MHz for the TSDF kernel has been achieved, resulting in total latency of 16.83 ms for the TSDF kernel and a total latency of 9.45 ms for the registration kernel. Considering the two main sections of the TSDF kernel, the TSDF projection and interpolation process has a latency of 6.8 ms, while a latency of 10 ms was estimated for the map merging part of the kernel. Hence, it can be observed that the calculation of the TSDF volume based on the incoming sensor data takes less time for computation than the update procedure, where the new volume is integrated into the already existing one. 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 that the system is able to track the position during the complete scanning process in real-time.

Table II shows the average runtime of the SLAM-Box and the Intel NUC while scanning an indoor environment. Our accelerated system works more than seven times faster than the NUC and handles incoming sensor data with more than 25 Hz, which is higher than the frequency of our laser scanner. Hence, received sensor data can be processed in real-time, enabling an accurate pose estimation during the complete scanning process, as discussed in the following section in more detail.

B. Accuracy

As mentioned in the previous section, the SLAM-Box allows the algorithm to localize itself with the maximum scan frequency of the used sensor. This minimizes the change of the pose between two time steps. As described in Section II, this is important for the accuracy of the algorithm because the conditions of the approximation of the Taylor series are better fulfilled. Since the matching procedure of our system can take place more than seven times faster than the unaccelerated version, the robustness of the localization and the accuracy of the pose estimation increase.

For evaluating the accuracy, we applied the SLAM-Box to different datasets recorded in indoor and outdoor environments. To evaluate the quality of the pose estimation, the system has returned to the previously marked start pose at the end of the scanning processes. Hence, we can compare the start and stop pose to estimate the correctness of the trajectory. Regarding all recorded datasets, we did not measure a distance exceeding $7.5\,cm$ using a map resolution of $6.4\,cm$ for every dimension. Hence, the accuracy is of the order of the discretization. The rotation varied by less than $2°$.

TABLE II: Comparison of performance, power and energy for the FPGA and the reference implementation

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

An additional scenario of a scanned indoor environment is depicted in Fig. 6, where an office building was scanned from the inside comprising a foyer, a part of the floor, and five rooms. In the left picture, the complete pose trajectory can be seen as a green path, and the turquoise to blue walls are the visualization of all scans recorded during the experiment. In the right picture, the created 3D mesh, generated based on the calculated TSDF data during the scanning process, can be seen from a similar perspective as the visualization of the pose trajectory. The SLAM-Box was able to track the pose during the complete procedure, resulting in a 3D reconstruction that fits the scanned environment.

C. Power consumption

To quantify the energy efficiency of the proposed implementation, we used a LowPowerLab Current Ranger in combination with an oscilloscope for power measurements. This enabled us to measure the current over time to trace the power consumption of individual processes. The power measurements include the FPGA board and the SSD; not included are the IMU (<0.3 W) and the LiDAR scanner (8 W typical). For this evaluation, a pre-recorded dataset was used, which was provided via Ethernet by a desktop computer. The results are shown in Fig. 7. The TSDF map update occurs roughly between t_1 and t_2. During this time, map update and registration are computed in parallel on the SLAM-Box, resulting in higher power consumption. Between the map updates, from t_2 to t_3, only the registrations are calculated. The individual registrations can be identified by dedicated peaks in the power consumption. The system has an average power consumption of $13.8\,W$ with lower and upper bounds of $12.36\,W$ and $15.24\,W$. Compared to the multithreaded software implementation on the NUC (cf. Table II), the FPGA-based platform requires less than half of the power while being able to process the data significantly faster. Overall, more than 18 times less energy is required to process one frame using the FPGA implementation compared to the reference system.

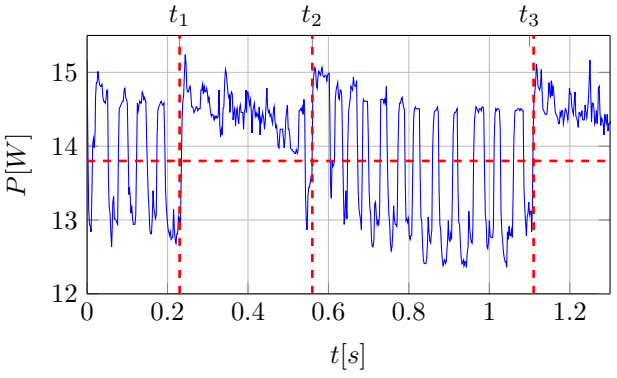

Fig. 7: The power consumption of the SLAM box over time, with the marked mean of $13.8\,W$

VI. Conclusion

An FPGA implementation of a LiDAR-based TSDF SLAM has been presented, targeting embedded real-time processing for applications with limited energy budget. The efficient combination of FPGA accelerators and embedded Arm processors on the used Xilinx UltraScale+ enables processing of all data at the maximum scanning rate of the used Velodyne VLP-16 sensor. Compared to a state-of-the-art PC, the developed SLAM-Box achieves a 7-fold speed-up and requires 18 times less energy. An optionally integrated ROS bridge enables online inspection of the mapping process as well as offline evaluation of pre-recorded data sets. The implementation provides accurate pose estimation with minimal drift. Thanks to its modular architecture, the approach can be easily adapted to new sensors or algorithmic modifications.

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. 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.

[4] 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, 2012.

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

[6] ——, "A Scalable FPGA-based Architecture for Depth Estimation in SLAM," in *ARC*, 2019.

[7] M. Belshaw and M. Greenspan, "A high speed iterative closest point tracker on an FPGA platform," *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, pp. 1449–1456, 2009.

[8] Q. Gautier, A. Shearer, J. Matai, D. Richmond, P. Meng, and 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.

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

[10] 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, 2021.

[11] S. Pütz, T. Wiemann, M. K. Piening, and J. Hertzberg, "Continuous shortest paths vector field navigation on 3d triangular meshes for mobile robots," in *Proceedings ICRA 2021*. IEEE, May 2021.

[12] 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.

[13] 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.

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

[15] Stanford Artificial Intelligence Laboratory et al., "Robotic operating system." [Online]. Available: https://www.ros.org