Redrose — Reconfigurable drone setup for resource-efficient SLAM

SEBASTIAN RAHN, PHILIPP GEHRICKE, CAN-LEON PETERMÖLLER, ERIC NEUMANN, PHILIPP SCHLINGE, LEON RABIUS, HENNING TERMÜHLEN, CHRISTOPHER SIEH, MARCO TASSEMEIER, THOMAS WIEMANN, and MARIO PORRMANN,
Osnabrück University, Germany

In this paper we present a heterogeneous architecture that integrates computing modules with FPGAs and GPUs into an existing UAV platform to allow real time TSDF-based SLAM directly on the drone. The system is fully integrated into the existing infrastructure to allow ground control to manage and monitor the data acquisition process. The system is evaluated in terms of power consumption and computing capabilities. The results show that the proposed architecture allows computations on the UAV that were previously only possible in post-processing while keeping the power consumption low enough to match the available flight time of the UAV.

CCS Concepts: • **Computer systems organization → Robotic autonomy**; *Reconfigurable computing*.

Additional Key Words and Phrases: ROS, FPGA, SoC, UGV, UAV, please update

ACM Reference Format:
Sebastian Rahn, Philipp Gehricke, Can-Leon Petermöller, Eric Neumann, Philipp Schlinge, Leon Rabius, Henning Termühlen, Christopher Sieh, Marco Tassemeier, Thomas Wiemann, and Mario Porrmann. 2018. Redrose — Reconfigurable drone setup for resource-efficient SLAM. In *DroneSE 2023: Drone Systems Engineering, January 18, 2023, Toulouse, France.* ACM, New York, NY, USA, 10 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

In previous work [4, 7], we presented an approach to run a hardware-accelerated TSDF-SLAM algorithm on an SoC with FPGA. It was tailored for a 32 line Velodyne VLP-32 LiDAR. Recent developments in LiDAR sensors now support up to 128 scan lines, increasing the required computational power to process all incoming data in real time. For applications in remote sensing, adding camera data to the acquired maps is desirable. However, this adds an additional modality to the sensor data stream which in turn again increases the computational load. In addition to FPGAs, SoCs with GPU accelerators offer high computational power with reasonable energy consumption. In this paper, we present a heterogeneous hardware architecture that combines the previously used SoC with FPGA accelerator with a GPU node in an fully integrated sensor and processing system on a UAV platform. The goal is to provide a SLAM system that computes a TSDF map on the fly while recording images from two high resolution RGB cameras. The system is fully integrated into the drone's control infrastructure, which allows to manage the data recording via ground control and send back monitoring information about the ongoing mapping process to the pilot.

DroneSE 2023, January 18, 2023, Toulouse, France

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

2 RELATED WORK

Heterogeneous system architectures for mobile systems are gaining attraction to allow computations on-board that have previously only been possible in offline processing. Especially in the area of mobile mapping, LiDAR-based SLAM systems with cameras are of importance. To generate high resolution maps from point cloud data, methods specially tailored for high frequency LiDAR data like LOAM [21], Lego-LOAM [13], LioSAM [14] and F-LOAM [15] have been developed to align point clouds in near real time on high-end CPUs. On UAVs, such algorithms are not yet real-time capable due to limited computing resources. In our previous work [3, 7], we developed a TSDF-based SLAM system that uses a SoC with FPGA to perform SLAM directly on a mobile platform. However, it is limited to 32 scan lines and scales not well to larger local maps. In the context of UAVs these local maps, which are used to align the data, have to be significantly larger because of the higher distance to the scanned surfaces. Due to memory bandwidth constraints, large local maps cannot be efficiently handled in FPGA-based systems. Given the experiences with other successful TSDF-SLAM approaches like KinectFusion [9] and Kintinuous [17], which exploit the massive parallel structures of GPUs, it is desirable to integrate such accelerators into the existing infrastructure on UAVs. In this paper, we describe a heterogeneous architecture that integrates different accelerators to address the TSDF SLAM problem on a drone. The workload of different parts of the developed algorithm is distributed to the different accelerator nodes running on the UAV. The computing nodes are fully integrated into the existing hardware of the UAV to allow control and monitoring of the data acquisition process on ground, while keeping the power consumption low, such that the possible mapping time matches the available flight time of the UAV.

3 DRONE SETUP

Our reconfigurable drone setup (Redrose) for resource-efficient SLAM can be mounted on the UAV platform and is based on two main processing boards, one equipped with FPGA and one with GPU. Besides these boards, two cameras and a LiDAR sensor are used to collect data.

3.1 UAV Platform

The UAV platform used to carry the system is a custom hexacopter built from off-the-shelf components. It is built around a Tarot PM X6 frame which has a motor-to-motor distance of 960 mm. Redrose including all its components is located below the battery tray for the UAV. On the stand, the frame has a ground clearance of 320 mm

Fig. 1. Photo of the UAV platform with Redrose.

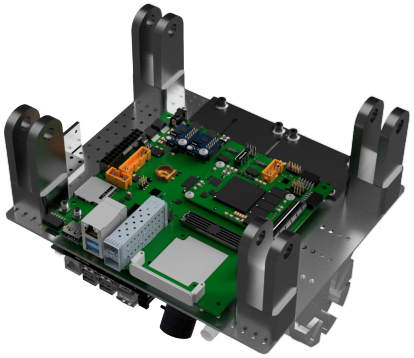

Fig. 2. 3D rendering of Redrose with the FPGA node mounted on the top of the setup.

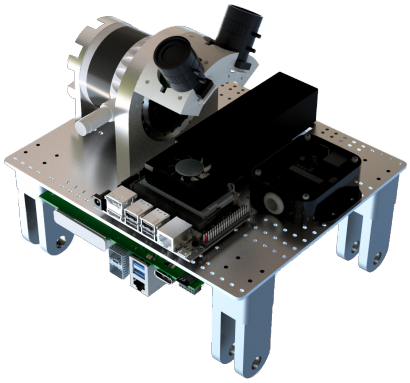

Fig. 3. Rendering of Redrose with LiDAR, cameras, GPU node and battery.

below the battery tray which is sufficient to hold all necessary components to perform TSDF SLAM. Additionally, the landing gear is electrically foldable after the takeoff so the data acquisition is not disturbed during a flight. The power supply for the UAV consists of two 10000 mAh 6s Li-Po batteries, which power the control electronic and motors of the UAV.

The control system relies on the CubePilot Cube Orange with ardupilot (V4.3) in conjunction with a Here3 GPS module and a herelink module for radio control, telemetry and video transmission. The UAV can be controlled manually with the herelink remote, but can also also perform missions prepared in ardupilot Mission Planer autonomously. Herelink features two video channels to send HDMI video feeds from the UAV to the remote. Channel 1 is used for an FPV camera to assist in controlling the UAV. The second channel transmits a live result preview of the SLAM setup. In addition to the herelink system, the drone also carries a RFDesign RFD868x-EU telemetry module to which a corresponding module on the ground can connect to. This allows a separate computers on the ground to connect to the UAV to control the compute modules added for SLAM.

Without Redrose, the UAV platform has a takeoff weight of around 7.6 kg and a payload capacity of about 5 kg. Including all sensors and compute modules, it has a takeoff weight of 11.3 kg. This allows a flight time of about 15 min, which enables the system to scan a large area before it has to land for a battery swap. The complete UAV platform including all parts for the TSDF SLAM is shown in Fig. 1. Fig 2 and Fig. 3 show renderings of the Redrose add-on components, which are described in the following section.

3.2 Redrose components

Redrose consists of multiple components integrated onto an aluminum plate, forming a single package. One part is the assembly that holds two cameras and an Ouster OS1-128 LiDAR sensor. As shown in Fig. 2 and 3, the two cameras are mounted at an angle of 60° and are mounted at the top of the LiDAR scanner.

Two compute modules, which form cooperating processing nodes, are screwed directly to the aluminum base plate and are connected to each other, the LiDAR sensor and cameras as well as the drone itself. Redrose uses an own Li-Po battery. Together with three separate adjustable voltage regulators to fit the different voltage requirements of the components, the battery provides power for the compute modules and the LiDAR. This setup is fixed to the carbon frame by several mounting brackets and weighs about 3.7 kg.

3.3 Sensors

Redrose consists of the LiDAR scanner and two 4K CSI cameras with 130° wide angle lenses. The LiDAR is an Ouster OS1 with a 45° vertical field of view and a resolution of 1024x128 points scanning at 20 Hz. It also integrates an inertial measurement unit (IMU), which is used for pose estimations in the SLAM algorithm.

Calibration. Due to the large field of view of the lenses, the images are heavily distorted. Hence, the cameras must be calibrated in order to match the pictures correctly to the scans. For this, several pictures from a chessboard of each camera from different angles and perspectives are undistorted using OpenCV to achieve a good calibration.

Co-Calibration. To be able to record color data for each laser scan, the correct offset between LiDAR scans and cameras must be determined accurately. We use the method described in [8] auto automatically compute the extrinsics without markers. For this, first the 3D scan is projected into a cylindrical image by transforming the world coordinates into camera coordinates. Then each point is projected to cylinder surface around the camera origin. In the last step the pixel coordinates for the resulting image are computed from the 2D coordinates. For these virtual scan images the reflectivity channel is used to computed grayscale pixel values. Because the camera has a smaller horizontal aperture angle compared to the LiDAR scan, only areas that are visible in both images only considered during calibration.

After that, the scan image and the camera image are compared with the Normalized Mutual Information (NMI) metric. For avoid local minima, the pictures are smoothed with a Gaussian Filter. To calculate the NMI score, a histogram of the intensity values is used as probability distribution. The intensity values are distributed into 16 histogram bins for better efficiency and smoother objective

function. Afterwards a Gaussian kernel is applied. In the last step, the objective function is optimized with the Nelder-Mead algorithm, starting with an initial guess from the Redrose model. The whole calibration is done with different scan/image pairs of a flight to achieve better calibration parameters.

3.4 Compute Modules

Apart from the LiDAR and the cameras, Redrose integrates two compute modules with different hardware architectures forming a heterogeneous computing platform. These computing nodes are the key components in the SLAM pipeline, capable of processing the camera and LiDAR scan data as well as controlling the application. One of these modules is an Multi Processor System on Chip (MPSoC) with a Zynq UltraScale+ Field Programmable Gate Array (FPGA) [5], which is integrated on a Trenz carrier-board [6]. This system forms the FPGA node used in our system and is shown in Figure 2. The FPGA node incorporates a 64-bit Quad-core ARM Cortex-A53 CPU as well as a Dual-core Arm Cortex-R5F co-processor among other processing units. Paired with 4GB DDR4 RAM and memory storage, the FPGA node enables the installation of PetaLinux, a Linux distribution used in systems utilizing Xilinx FPGAs. This allows the interfaces important for Redrose, such as USB3.0, Ethernet and General Purpose Input/Output (GPIO), to be configured and used. The FPGA node, which is used for preprocessing of the laser scan data has programmable hardware, which can be adapted to our algorithms and therefore run more efficient and perform better than simple software implementations on conventional systems utilizing only CPUs.

The second compute module incorporated in the setup is the Jetson Xavier NX [2] which runs on Ubuntu Linux and is shown in Figure 3 among other components. This compute module forms the GPU node in Redrose and features an embedded GPU alongside its 6-core ARM CPU. Since GPUs shares a massively parallel processor architecture by design, they can accelerate tasks where it is necessary to execute the same instructions on many objects from large datasets. One field of use is to perform calculations for the pixels of an image in parallel instead of doing so sequentially. Furthermore, the GPU node takes advantage of hardware encoder and decoder to efficiently process image data from both cameras, which are connected to the GPU node via CSI. The 8GB LPDDR4x memory is shared by the CPU and GPU and stores, among others, the SLAM data. To store the map, scan and image data, an Solid State Storage (SSD) is connected to the GPU node. Using the GPIO pins, the GPU node can receive and send commands from and to the ground control. Furthermore, the GPU node sends information to the ground control via HDMI, which is displayed directly on the remote control. Details of the ground control are described in Section 4.4 and 5.3.

The tight integration of the different hardware architectures forms a heterogeneous platform, using a communication infrastructure which is also shared by the LiDAR and cameras, allowing to exploit the advantages of the two hardware architectures respectively.

4 HETEROGENEOUS HARDWARE ARCHITECTURE

This section presents the heterogeneous hardware architecture and its components. The defined communication interfaces between the individual, the top level data flow and the components are also discussed.

4.1 Communication

Interfaces. An abstracted overview of the main communication interfaces and the most important tasks are presented in Fig 4. Three of the four main components, namely the LiDAR scanner, FPGA and GPU node, communicate with each other via transmission control protocol/internet protocol (TCP/IP) over Ethernet interfaces. Ethernet was chosen as the interface because it provides sufficient bandwidth with 1 Gbit/s, drivers and a stable connection. In order to enable a direct communication between the GPU node and the LiDAR scanner, a network bridge is configured on the FPGA node. To simplify development and debugging, data between the FPGA and the GPU node can be retrieved. This configuration is called development mode and can directly be used to visualize and analyze the data.

In order to send commands to Redrose, the ground control transmits the corresponding signals to the orange cube, which then generates pulse width modulation signals (PWM) that are read on the GPIO pins of the GPU node. Furthermore, an HDMI stream is sent from the GPU node to the UAV, which then transmits it via a 2.4 Ghz WiFi channel to the remote control. The MAVLink commands to control the UAV are also transmitted over this channel.

Dataflow. The LiDAR scanner send point clouds and IMU data via Ethernet to the FPGA node. Since the amount of data for the point cloud and IMU files is less than 5 MB, the latency that occurs during transmission can be neglected. As soon as the ARM CPU on the FPGA node receives the data, it is written to the DRAM. When the kernels on the FPGA node have finished processing the data, the ARM CPU sends the processed IMU and point cloud data to the GPU node via Ethernet. The amount of data has become smaller due to the processing, so that the transmission latency is also negligible Here the data is received from the bridges, which are managed by the CPU on the GPU node, and processed by the SLAM callback on the GPU. The camera data is processed on the GPU in parallel and then streamed to the UAV via HDMI and stored on the SSD.The UAV finally forwards the data via a 2.4 Ghz WiFi channel to the remote control, on which the HDMI signal is displayed.

4.2 FPGA-Accelerator

FPGAs can make a big improvement when it comes to processing a large amount data concurrently. We take advantage of this to pre-process the laser scans efficiently and quickly. To work with the FPGA accelerator, Xilinx Design Flow version 2021.2 is used [20]. This process is implemented through a scripting approach to simplify execution and customization. The design consists of four basic building blocks, the hardware design, the operating system, the kernels and the software application. In the following, these parts are discussed in more detail.

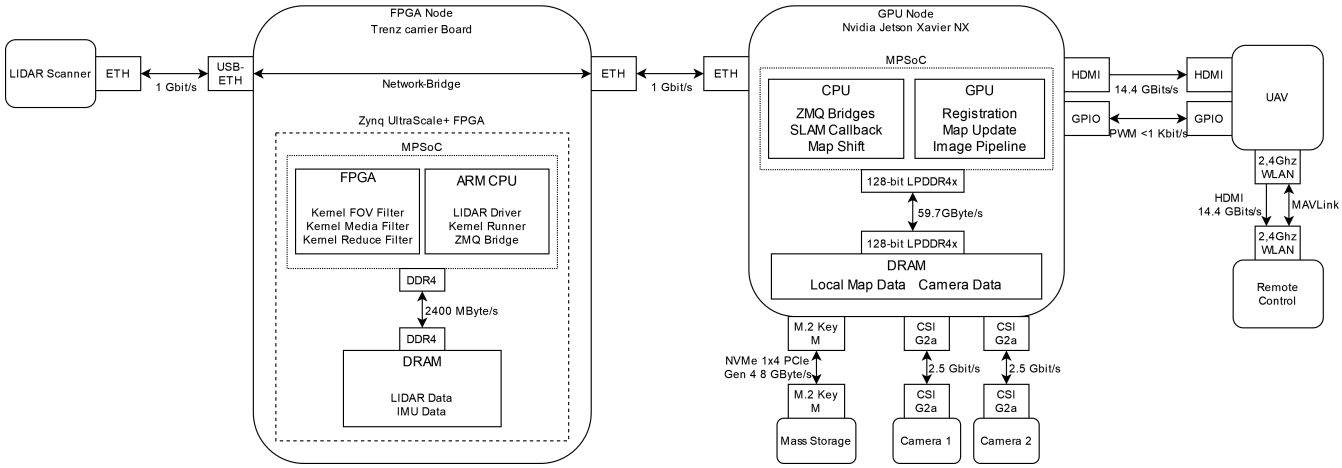

Fig. 4. Network architecture including the used interfaces (shown as rectangles) and their theoretical maximum data rates. In addition, the most important tasks for each part of the MPSoCs are indicated and which data the task accesses in the respective DRAM.

Hardware design. During the hardware design, the interfaces of the individual components are defined. The processing unit provides clocks for the FPGA and transfers data via AXI interfaces. Another important aspect is the connection to the baseboard, which has to be implemented in order to use the interfaces such as GPIO and HDMI. In addition, configurations can be made to the processing unit. Settings such as memory interfaces, address mapping and clock frequencies are changed. With the hardware design as the foundation, the next step is to build the operating system.

Operating system. According to the design flow provided by Xilinx, a Linux-based operating system called Petalinux is used [18]. Once created a new project, a minimal operating system is initially created based on the hardware design. This provides a lean command line system with basic Linux commands and drivers, which can also be thoroughly customized to suit individual needs. Components like the device tree, Linux kernel and rootfs can therefore be customized. An extensive catalog of settings, packages, libraries and applications is offered. Moreover, software can be cross-compiled with the help of bitbake directly into the system. To set up the operating system for our use case, several components are added, which are listed in Tab. 1.

After the compilation process of the project, a bootable image is created which then runs on the FPGA board. That platform can now be used to utilize the resources of the FPGA and accelerate an application with embedded kernels.

Kernels. To embed an application that accelerates the pre-processing of the point clouds on the FPGA, the Vitis Kernel Flow is used [19]. This approach includes writing kernel code to accelerate the algorithms, which are implemented in C++ and translated via the Vitis tool into hardware for the FPGA. In order to accommodate hardware-specific optimizations, pragmas are offered, which allow for code-specific pipelining, memory handling, and other optimizations. Two of these optimizations were implemented in the kernel, to ensure an optimal runtime: pipelining and memory optimizations.

Table 1. Packages cross-compiled into the Petalinux operating system.

Package	Description
pcl	Library for Pointcloud processing
libgpiod	Library to access GPIO ports
zeromq	Network messaging library used to communicate with the GPU node
USB RTL8152	USB-to-Ethernet Adapter driver
vim	User-friendly and efficient text editor
iperf	Measures the maximum bandwidth on IP networks
ntp	Synchronizes the time between the FPGA- and GPU-Node
bridge-utils	Utility to create and manage bridge devices
init-ifupdown	Setting up the default network configuration
initsdcard	Automatically executes a script at startup. Used to setup ntp, the bridge and start the main application

Another requirement is that a host code must be written to implement the interface between the CPU and the FPGA kernel. The point cloud and additional metadata are transferred via this to the kernel code. Besides that, configuration files can be used to adjust the number of kernels and their memory interfaces. After the implementation of kernel and host code, they are added to the previous base design. This will create a bootable image containing files to run the application for accelerating the pre-processing of the point clouds on the FPGA.

Software. The host code runs across three main threads. One of these threads handles the data from the laser scanner, the other communicates to the FPGA and the last one sends data over a bridge to the GPU node. Apart from that, direct configurations are made by the operating system itself to implement the network handling and especially setting up a network bridge between the LiDAR scanner

and the GPU node. The FPGA node can be started via the drone control and thus the reading and pre-processing of the laser scan data.

4.3 GPU-Accelerator

GPUs are built to handle massively parallel work. While the FPGA can only use statically sized memory and prefers integer computations, GPUs excel at applications that operate with dynamically allocated memory and floating point arithmetic. With general-purpose GPU (GPGPU) APIs like CUDA, we can make use of this architecture to accelerate processes while staying close to CPU code. To utilize the GPU, data streams need to be managed in order to communicate with the different computation nodes.

SLAM. The network communication between the GPU and FPGA node is implemented with the zeromq library. The FPGA node opens two separate network sockets, which are accessed by the bridges on the GPU node (see Fig. 4). Communication between a bridge and the SLAM callback thread is implemented using two buffers for IMU and LiDAR data, which follow a queue principle (FILO). If the SLAM algorithm cannot process the data at the intended frequency, the old messages are discarded so that only recent scans are used. After a scan is fetched from the buffer, the SLAM callback copies it into the shared memory and executes the registration and, if necessary, performs a map update afterwards. To achieve efficient data access, we are using a swapping strategy, where a part of the local map is held in shared memory and the global map is stored on the SSD. After registration, the updated drone position is provided to the map shift thread, where a shift is performed if the traveled distance exceeds a threshold. Here, the required areas are loaded from the SSD into DRAM memory before the map shift interrupts the SLAM callback and updates the local map.

Image processing. Computation power on the FPGA node is not sufficient for processing two parallel uncompressed video streams and saving them afterwards. Therefore we use a direct connection from the cameras to the GPU accelerator. As many parts as possible are computed inside the GPU managed memory with CUDA kernels. After getting the raw sensor data, a debayer as well as color conversion kernel is applied to the images which are undistorted afterwards via lookup tables derived from the calibration described previously. If the following pipeline is limited by resolution, we can also apply a resize kernel to reduce data rates. Using CUDA streams, all image operations can be pipelined and executed efficiently before further use. To increase efficiency and resource usage in the image processing pipeline, we make use of the nvJPEG library, which utilizes hardware encoders to convert JPEG images. This reduces the load on the CPU when saving files to the disk, since preparing the buffer, reading the resulting buffer and handling the file system is omitted. Processing the images to JPEG is not mandatory, but saves memory on the SSD and decreases the I/O operations, which in turn reduces the load on the CPU.

On the CPU, a C++ application with an event-driven approach is used. Each step of the processing pipeline is realized by a thread containing an event queue. This opens the opportunity to utilize every core of the CPU and avoid blocking events because of expensive I/O operations. Every component can subscribe to the desired events, which are stored in the event queue. Enabling and disabling components on demand and also intercepting events allow to create metrics or display debug information via the viewer of the ground control.

4.4 Ground Control

The ground control monitors and controls the algorithms on our nodes during flight. For signal transmission to the drone pyMAVlink, a communication protocol for drones and drone components, is utilized [10]. The debug viewer sends current camera images to the drone's remote control via the HDMI downstream which assists monitoring Redrose.

With a long range antenna, a PC sends commands in form of MAVLink messages, which are received by the Orange Cube and used to control up to six servo ports. These ports function as communication interfaces for the GPU node by using pyMAVlink commands, which generate PWM signals on each port separately, resulting in messages in six-bit format. These messages are read through the GPIO pins of the GPU node and are converted to a binary representation. The GPIO pins are constantly monitored and each servo port is initially set to zero.

An example of how a message is transmitted is shown in Fig. 5. A message transmission starts, when a positive edge is detected on any GPIO pin. The message protocol works in such a way, that all positive edges of each pin are detected first and then all negative edges are read. For each positive edge detected on a GPIO pin, its associated complement is added to a global counter. Furthermore, the order in which the negative edges were detected on the pins must be the same as the order in which the positive edges were detected. If the servo ports are deactivated in a different order, the message is considered invalid and has to be sent again. If no errors are detected in the sequence, the message is accepted and the command with the respective counter coding is executed. The repertoire of functions that can be executed include the acquisition for LiDAR and camera data, controlling the power controls of the FPGA node and rebooting the GPU node.

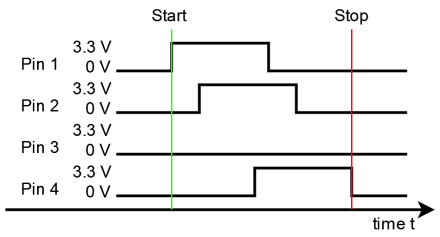

Fig. 5. Example transmission with four pins in use.

5 IMPLEMENTATION

This section details the implementation of the different appilcations running in Redrose described in the previous section.

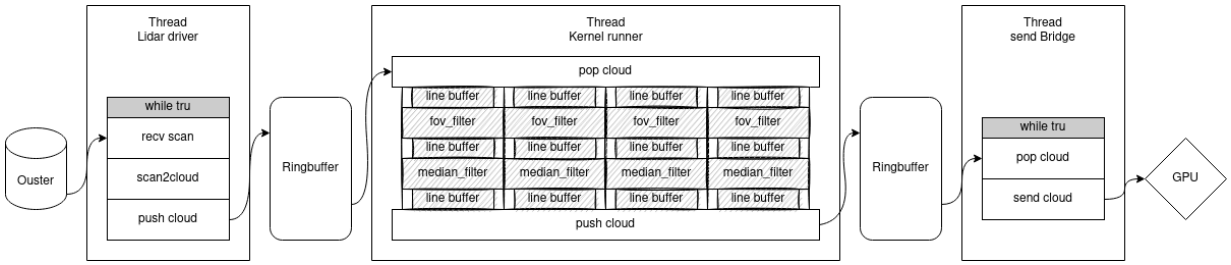

Fig. 6. A visual representation of the data flow on the FPGA. The scans are received from the LiDAR and passed to three threads: LiDAR driver, Kernel runner and send bridge. At the end, point clouds are sent to the GPU node.

5.1 FPGA Application

The FPGA application consists of three threads that are connected in a pipeline with each thread working independently. These threads are the LiDAR driver, the kernel runner, and the send bridge, depicted in Fig. 6.

LiDAR Thread. The pipeline starts with the LiDAR driver, which continuously receives new data from the laser scanner. Subsequently, the individual recordings are converted into 3D data to be stored in a point cloud data structure. These 3D points are converted to an integer representation, since integers are processed more efficiently than floating-point numbers on an FPGA. The Ouster LiDAR generates 128×1024 points, which are received every 50 ms. At the end, the pointcloud is pushed into a ring buffer, from where it is processed further by the next thread.

Kernel Thread. The kernel thread starts by reading the point cloud from the ring buffer. Since we have four kernels, we divide the point cloud into four equal parts, one for each kernel capable of processing the point cloud in parallel. At the beginning of each kernel, a complete ring of 1024 points is loaded into the local memory of the respective kernel as a buffer which can be seen in Fig. 6. This allows pipelining and massive parallel access to the individual elements of the buffer, without the need for global memory access. The first stage of pre-processing involves a filter that excludes points that are not within a specified field of view, called FoV_filter. This field of view is defined by a start angle and an end angle, between which the remaining points should lie. Points that lie outside these boundaries are set to zero. After the field of view filter the kernel continues with a median filter which averages the point cloud to remove outliers. In order to implement this, a sliding window with five elements is moved over a scan line, and the center points of the window are replaced with the average value of the elements. Zero points are not considered to prevent data from being distorted. When the kernel thread ends, the buffers of each kernel are combined into a point cloud and pushed into the ring buffer.

Bridge Thread. After the kernel thread has finished, the pre-processed point cloud is sent to the GPU node. To minimize transmission bandwidth, points with zero values can be omitted. For transmission, ZeroMQ is used to send data to the GPU node via an Ethernet connection, where it is further processed. In development mode, these transmissions can be received by another computer, to visualize the point cloud. Besides that, the bridge also sends IMU data from the LiDAR directly to the GPU node.

5.2 GPU Application

SLAM. Simultaneous Localization and Mapping (SLAM) is a fundamental problem in robotics, which is well discussed in the literature. Many approaches exist to solve this problem for 3D maps based on LiDAR data. Our work uses the methods and implementations from [4], which have been slightly modified to run on the embedded GPU used in Redrose. These methods are based on an incremental localization algorithm that uses a Truncated Signed Distance Field (TSDF) as a map representation. For efficient computation, a swapping strategy is used, where the map region around the current pose is kept as a local map in GPU memory. If the traveled distance exceeds a specified threshold, the areas outside this fixed region are integrated into the global map and the empty regions are filled with existing data. The global map itself is stored on disk to allow scanning of large areas.

The main reason for using the GPU for SLAM is the increased memory bandwidth, which is a bottleneck in the FPGA approach [4]. Moreover, the map update is expected to be accelerated due to the massively parallel computation on the GPU.Furthermore, we execute the registration and map update in succession 7. Previously, the available processing power of the FPGA was divided between map update and registration, which made it necessary to run the kernels in parallel for optimal usage. This resulted in additional synchronization steps that interrupted the processing pipeline and caused idle times. By using the GPU, the registration and map update can now use the full capacity of the provided hardware individually. As a result, we are not faced with performance losses due to additional synchronization steps.

Map Update. The used TSDF map is a 3D voxel grid that contains an implicit representation of the surface. Each voxel stores the distance to the surface and a weight that encodes the certainty of the distance value. To integrate a scan into the map, a ray-marching approach based on the methods in [1] is used to compute a temporal TSDF volume. Each ray is traversed from its scan point to the computed pose from the registration step. The distance value of intersected voxels are then updated with the current distance to the respective point. Since multiple rays can pass through a voxel, only the smallest distance to the surface is kept. Finally, a weighted average procedure is used to integrate the temporal map into the local map.

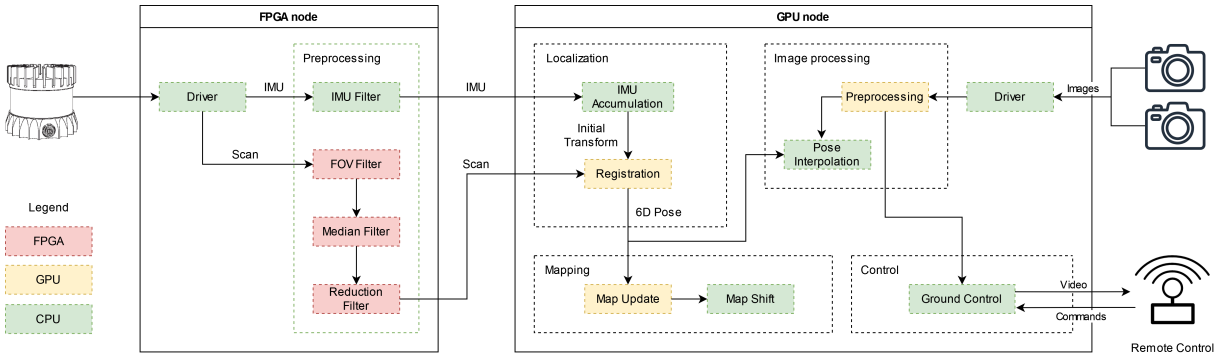

Fig. 7. Dataflow between processing steps of Redrose. LiDAR and IMU messages are pipelined through the FPGA and GPU node. Before the images are sent to the remote control, they are processed and labeled with their poses.

Registration. For the registration of a scan, a Point-to-TSDF method from [1] is applied iteratively to compute a transformation matrix. This method defines the registration as a minimization problem, in which the error of a computed transformation can be understood as the deviation between points and the actual surface and is therefore defined as the sum of the TSDF entries over all points. The distance values of the voxels around a point define a gradient, which is used to calculate a transformation that moves the points in the direction of the surface. The iterative application of this method improves the transformation until an error threshold or a maximum number of iterations is met. To stabilize the calculation of the transformation matrix, an additional weight is used, which grows linearly over the number of iterations and reduces the influence of individual transformation updates. An initial pose estimation for this algorithm is provided by accumulating IMU measurements between two consecutive scans as shown in Fig. 7.

Image processing. GPUs were invented to accelerate processing pixel data. Since we have 2 camera streams in parallel, we offload operations to the GPU and minimize memory operations to reduce latency. It starts with getting the image. Since capturing is a blocking call, there is one thread per camera. Each thread opens a video stream with the NVIDIA Argus API [12] to support a copy-free pipeline for the upcoming processing on the GPU. All described image operations are computed with CUDA in GPU memory and the result is distributed as a pointer to GPU memory via the event system. The image gets timestamped and runs through a debayering kernel to retrieve the full colored image. Afterwards the image is converted to RGB format and shrinked to a resolution of 1920 x 1080 pixel using a resize kernel.

The poses of the registration to the corresponding images are added as metadata for future processing. To improve the pose estimation linear interpolation of the registered poses is used by utilizing the timestamps of the laser scans and images. If the required poses are not determined yet, the image remains in the GPU memory until the interpolated pose can be obtained. In the last processing step the images are encoded to JPEG, copied to the CPU managed memory and saved on the carried SSD.

To encode the images to JPEG, the embedded hardware encoder on the GPU node is utilized. It reaches 460 MHz, which enables encoding of 30 images per second with a resolution of 1920 x 1080 pixel. There are CPU implementations like libjpeg-turbo which aim to increase throughput by implementing intrinsic functions of the CPU architecture but only encode two images per second on our GPU node. GPU implementations also exist [16], but they come with the cost of more GPU usage which is already prioritized for SLAM. We therefore use the resource efficient hardware encoder and encode two camera streams in ten frames per second each.

5.3 Ground Control

While the drone is in the air, the pilot is focused on flying. The Ground Control assists the pilot and other operators on the ground controlling the Redrose application on the drone.

Viewer. The remote control has the capabilities to show a video feed to the pilot via HDMI Downlink. In Fig. 8, the rendered output sent by the drone to the remote control is shown. The background of the transmitted feed is a video stream of one of the cameras. On top of it, an overlay displays relevant system information. This viewer runs on the GPU node and retrieves all images through the event system while ignoring images with a configurable delta time from the image timestamp to ensure a recent stream for the pilot. The viewer process runs an http server which can be accessed from the network by every running process with the capability to open TCP connections. This way the viewer collects selected state changes and other metrics like battery voltage from the system and presents them to the pilot. The API is also used to dynamically select which camera is shown to the pilot. Before streaming the image with HDMI Downlink, it is firstly color converted to RGBA and then alpha blending is applied with the generated overlay image. The result is then sent to nvdrmvideosink from NVIDIA with GStreamer [11] that directly writes to the HDMI output of the Jetson which is connected to the Downlink antenna of the UAV. This way we save CPU load and memory consumption for running a Linux window server.

Webinterface. To ensure easy operation and control of the GPU and FPGA node on the drone, we developed a web interface. The integration of pyMAVLink is made possible by the fact that the web interface is based on Python. Figure 9 illustrates the main structure of the website. At the top, there are configuration options for the

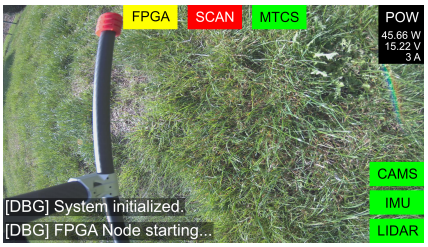

Fig. 8. Monitoring the system state on the remote control display streamed with HDMI Downlink.

connection to the drone. This includes the number of servo ports used for message transmission, the connection interface of the long-range antenna and the BAUD rate. Subsequently, the connection can be created by clicking on the "Create Connection" button. If the connection is successful, more buttons for sending messages will appear. These buttons enable you to perform certain functions, like increasing or decreasing the time for test data recordings or starting them. For debugging purposes, there is the possibility to transmit a self-defined bit order.

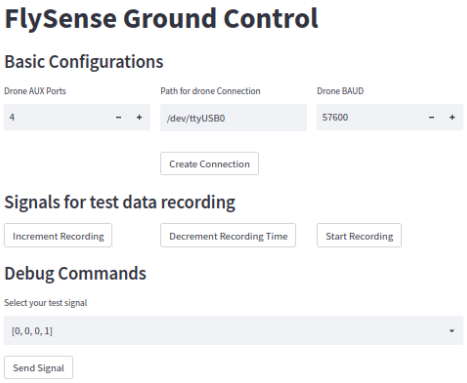

Fig. 9. The Ground Control GUI allows you to connect and send messages to the drone while it is in the air.

6 EVALUATION

For the Redrose system, it is essential to run applications as efficiently as possible on the low-power hardware. To analyze this, the runtime of the algorithms and the utilization of the hardware are considered. The FPGA node, the GPU node and the general power consumption of Redrose will be discussed.

6.1 FPGA node

In the following section, the application running on the FPGA node is analyzed by measuring the performance and resource utilization. For this purpose, the kernel thread responsible for the main processing of the pointclouds is considered. The measurement data were collected for one, two, four and eight kernels.

Measurements were made for the kernel and the total thread time, which are illustrated in Tab. 2. The kernel time considers how long it takes to process a point cloud on the FPGA hardware. The total thread time is the duration for the pipeline to process one iteration, including the kernel time, communication with the FPGA and reading/writing the ring buffers. Increasing the number of kernels generally results in faster runtimes. On average, increasing the number of kernels from one to four makes the execution time 40 % faster. When the amount of kernels increases from one to two, the total time increases by 41.2 %, while it only increases by 15 % from two to four. The speed boost results from the fact, that kernels run concurrently. Each of them has its own buffer, which results in massive parallel data processing.

Number of kernels	task	min	max	avg
1x	total	33 ms	39 ms	34 ms
	kernel	14 ms	15 ms	14 ms
2x	total	16 ms	33 ms	20 ms
	kernel	8 ms	11 ms	8 ms
4x	total	16 ms	17 ms	17 ms
	kernel	5 ms	5 ms	5 ms
8x	total	13 ms	18 ms	17 ms
	kernel	4 ms	6 ms	6 ms

Table 2. Execution time of the kernel. In addition, different numbers of kernels were used.

Looking at the utilization of FPGA resources in shown in Tab. 3, the hardware usage doubles as the number of kernels increase. In general, a single kernel on the FPGA does not use many resources. If the number is increased to four, still ≈80 % of the FPGA is available for other tasks. With more than four kernels, the pre-processing does not become faster at all, but instead just unnecessarily takes up more resources. This can be explained by the fully loaded memory interface, which does not have more than six connections. Thus further kernels are slowed down.

Table 3. FPGA resource utilization for different number of kernels.

Number of kernels	CLB	LUTS	BRAM
1x	7.35 %	4.36 %	4.57 %
2x	12.67 %	8.27 %	9.41 %
4x	23.69 %	15.62 %	18.28 %
8x	47.51 %	31.27 %	36.56 %

6.2 GPU node

SLAM performance. To evaluate our SLAM algorithm, we measured the computation time required for registration and map update. For this purpose, we created a network recording of the LiDAR, which was then converted into a ROS Bag file to use the extensive functions of the ROS environment. A ROS interface allows us to stream the data from recordings back into our heterogeneous system and monitor the processed scans and generated TSDF map, see Fig. 10 above. The recording contains 1236 scans with a resolution of 1024x128 points at a frequency of 20 Hz. The resulting times of

the SLAM algorithm can be seen in the diagram in Fig. 10. Here, the "total" category includes the times for registration and map update, as well as a previous type conversion and copy of a scan into the GPU shared memory. Despite individual outliers, the duration that our SLAM algorithm needs for a scan is in the range of 50-100 ms. It should be emphasized that the map update has received a speed-up by an order of magnitude compared to [3], due to the possibility of a massively parallel computation of the simple ray-marching algorithm on the GPU. With a maximum duration of 2 ms, the influence on the total time is marginal. An overview of the evaluation is given in Tab. 4. Qualitatively, due to the large measurement ranges of our application, even small registration errors cause visible drifts in the outer areas of the local map, which are visible in Fig. 10 in the left panel of the map.

Table 4. Runtimes of registration and map update step on the GPU node. The "Total" category includes the times for registration and map update plus type conversion and copying into the GPU shared memory.

task	min	max	average
Total	9 ms	253 ms	55 ms
Registration	3 ms	143 ms	44 ms
Map Update	0.2 ms	2 ms	0.4 ms

Image processing. To examine the impact of the image processing on the GPU node, we measured CPU and GPU utilization as well as memory usage. In Tab. 5, the utilization is listed for image processing only. It should be noted, that the load of the operating system with GUI is included in the measurements. We observed that the CPU utilization is a little more than a third and the GPU is utilized about a quarter on average. However, the peak loads of the GPU should not be underestimated, where half of the GPU is needed for image processing. We assume that this due to interference with SLAM process. Especially in time critical tasks this could lead to problems, for example during registration.

Table 5. GPU node utilization for image processing without viewer display.

Component	min	max	average
CPU	32 %	44 %	≈38 %
GPU	0 %	56 %	≈25 %
Memory	3.6 GB	3.7 GB	≈3.7 GB

Viewer. In addition to the images from the image processing, the viewer adds information to the frames and outputs them via HDMI. This additional load is shown in Tab. 6. Not only the CPU and GPU load increases by 5-7 %, but also 500 MB more memory is needed. Due to the integration of the viewer in the image processing pipeline, the difference is fairly low compared to the rest of the application.

6.3 Energy Consumption

An important requirement for Redrose is a low power consumption. Since the drone has a flight time of about 15 minutes, the additional components and modules for the SLAM should be able to record data and perform the according algorithms for the same duration. For the components of Redrose a 74 Wh battery is used. The separate voltage regulators are set to 19V for the GPU node, 12V for the FPGA node and 24V for the laser scanner. The measured power consumption of the setup results in around 40 W at idle without running SLAM. During idle, separate measurements show a power consumption of 7 W on the GPU node, about 16 W on the FPGA node and 16 W on the LiDAR scanner.

Table 6. GPU node utilization for image processing with viewer display.

Component	min	max	average
CPU	39 %	56 %	≈45 %
GPU	0 %	66 %	≈30 %
Memory	4.1 GB	4.2 GB	≈4.2 GB

Due to the typical characteristics of FPGAs, the FPGA node in our configuration consumes about the same amount of power under load as running on idle. For this reason, the measured power consumption of the FPGA node while performing the kernels is again about 16 W. The GPU node on the other hand consumes about 19 W running all necessary steps of SLAM. This results in a total power consumption of around 52 W under full load. Without discharging the 74 Wh battery too low, it can supply Redrose with energy longer than an hour and therefore can last for multiple flight sessions assuming additional sets of batteries are provided for the UAV itself.

7 CONCLUSION AND FUTURE WORK

A heterogeneous processing platform for drones has been presented that seamlessly integrates embedded processors, reconfigurable hardware and GPUs in a compact yet powerful environment. The combination of FPGA-based and GPU-based accelerators facilitates a wide variety of options for hardware software partitioning. Tightly integrating the processing environment into the control infrastructure of the drone combined with a sophisticated monitoring environment eases the development and evaluation of new applications.

The first target application that utilizes the heterogeneous architecture is an integrated SLAM system that combines on the fly computation of a TSDF map with high-resolution image processing. The heterogeneous platform enables the integration of a LiDAR sensor with 128 scan lines, significantly increasing the resolution but also the computational requirements compared to previous work [4]. The application has been partitioned to the embedded processors, GPU and FPGA fabric targeting real-time performance with minimum energy requirements. First results are highly encouraging, indicating that online processing of the complete application is possible on the heterogeneous platform without reducing the flight time of the UAV.

Future work concentrates on optimizations of the approach to further increase accuracy, performance and energy. In addition to low-level optimizations of the FPGA and GPU implementation, new data structures like octrees and hashmaps will be evaluated to circumvent the current memory bottlenecks.

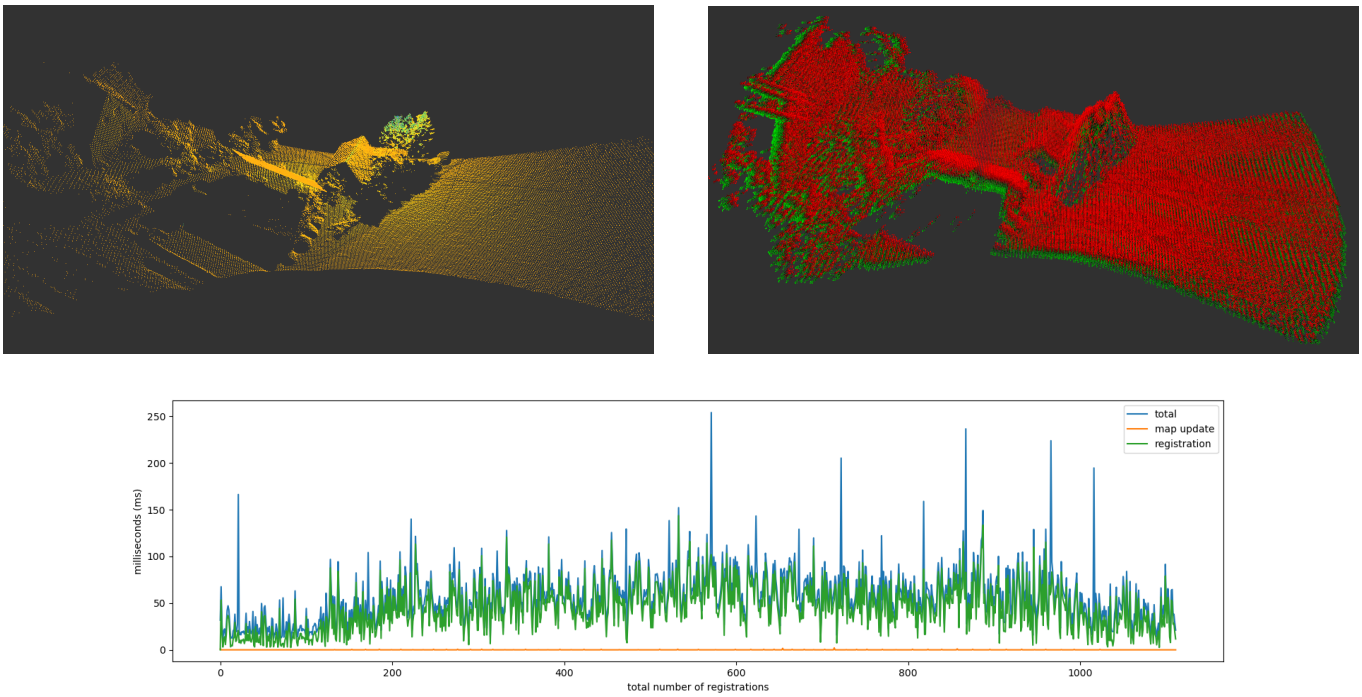

Fig. 10. Top: A drone scan of a rural area, with the corresponding TSDF local map. Bottom: Runtime evaluation of our slam algorithm on on the GPU node based on a recording of 1236 scans with a resolution of 1024x128 points at 20 Hz

REFERENCES

[1] Daniel Ricao Canelhas, Todor Stoyanov, and Achim J. Lilienthal. 2013. SDF Tracker: A parallel algorithm for on-line pose estimation and scene reconstruction from depth images. *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2013), 3671–3676.

[2] NVIDIA Corporation. 2022. Jetson Xavier NX Series. https://www.nvidia.com/de-de/autonomous-machines/embedded-systems/jetson-xavier-nx/. Accessed: 2022-11-22.

[3] Marc Eisoldt, Marcel Flottmann, Julian Gaal, Pascal Buschermöhle, Steffen Hinderink, Malte Hillmann, Adrian Nitschmann, Patrick Hoffmann, Thomas Wiemann, and Mario Porrmann. 2021. HATSDF SLAM–Hardware-accelerated TSDF SLAM for Reconfigurable SoCs. In *2021 European Conference on Mobile Robots (ECMR)*. IEEE, Bonn, Germany, 1–7. https://doi.org/10.1109/ECMR50962.2021.9568815

[4] Marc Eisoldt, Julian Gaal, Thomas Wiemann, Marcel Flottmann, Marc Rothmann, Marco Tassemeier, and Mario Porrmann. 2022. A fully integrated system for hardware-accelerated TSDF SLAM with LiDAR sensors (HATSDF SLAM). *Robotics and Autonomous Systems* 156 (2022), 104205. https://doi.org/10.1016/j.robot.2022.104205

[5] Trenz Electronic. 2022. Trenz Electronic Wiki - TE0808 TRM. https://wiki.trenz-electronic.de/display/PD/TE0808+Resources. Accessed: 2022-11-22.

[6] Trenz Electronic. 2022. Trenz Electronic Wiki - TEBF0808 TRM. https://wiki.trenz-electronic.de/display/PD/TEBF0808+TRM. Accessed: 2022-11-22.

[7] Marcel Flottmann, Marc Eisoldt, Julian Gaal, Marc Rothmann, Marco Tassemeier, Thomas Wiemann, and Mario Porrmann. 2021. Energy-efficient FPGA-accelerated LiDAR-based SLAM for embedded robotics. In *2021 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, Auckland, New Zealand, 1–6. https://doi.org/10.1109/ICFPT52863.2021.9609934

[8] Felix Igelbrink, Thomas Wiemann, Sebastian Pütz, and Joachim Hertzberg. 2019. Markerless AD-HOC calibration of a hyperspectral camera and a 3D laser scanner, In Intelligent Autonomous Systems 15. *Advances in Intelligent Systems and Computing* 867, 748–759. https://doi.org/10.1007/978-3-030-01370-7_58

[9] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, et al. 2011. KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*. 559–568.

[10] MAVLink. 2022. MAVLink User Guide. https://mavlink.io/en/. Accessed: 2022-11-24.

[11] NVIDIA. 2019. Accelerated GStreamer User Guide. https://developer.download.nvidia.com/embedded/L4T/r32_Release_v1.0/Docs/Accelerated_GStreamer_User_Guide.pdf. Accessed: 2022-11-25.

[12] NVIDIA. 2022. *Libargus Camera API.* Retrieved 2022-11-25 from https://docs.nvidia.com/jetson/l4t-multimedia/group__LibargusAPI.html

[13] Tixiao Shan and Brendan Englot. 2018. 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)*. IEEE, 4758–4765.

[14] Tixiao Shan, Brendan Englot, Drew Meyers, Wei Wang, Carlo Ratti, and Daniela Rus. 2020. Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping. In *2020 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 5135–5142.

[15] Han Wang, Chen Wang, Chun-Lin Chen, and Lihua Xie. 2021. F-loam: Fast lidar odometry and mapping. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 4390–4396.

[16] André Weißenberger and Bertil Schmidt. 2021. Accelerating JPEG Decompression on GPUs. In *2021 IEEE 28th International Conference on High Performance Computing, Data, and Analytics (HiPC)*. 121–130. https://doi.org/10.1109/HiPC53243.2021.00026

[17] Thomas Whelan, Michael Kaess, Maurice Fallon, Hordur Johannsson, John Leonard, and John McDonald. 2012. Kintinuous: Spatially extended kinectfusion. (2012).

[18] Xilinx. 2021. *PetaLinux Tools Documentation: Reference Guide* (ug1144 (v2021.2) ed.). Retrieved 2022-11-26 from https://docs.xilinx.com/r/2021.2-English/ug1144-petalinux-tools-reference-guide/Revision-History

[19] Xilinx. 2021. *Vitis High-Level Synthesis User Guide* (ug1399 (v2021.2) ed.). https://docs.xilinx.com/r/2021.2-English/ug1399-vitis-hls/Getting-Started-with-Vitis-HLS

[20] Xilinx. 2021. *Vivado Design Suite User Guide: Design Flows Overview* (ug892 (v2021.2) ed.). Retrieved 2022-11-26 from https://docs.xilinx.com/r/2021.2-English/ug892-vivado-design-flows-overview/Vivado-System-Level-Design-Flows

[21] Ji Zhang and Sanjiv Singh. 2014. LOAM: Lidar odometry and mapping in real-time.. In *Robotics: Science and Systems*, Vol. 2. Berkeley, CA, 1–9.