ReconfROS: Running ROS on Reconfigurable SoCs

MARC EISOLDT, MARCEL FLOTTMANN, JULIAN GAAL, STEFFEN HINDERINK, JURI VANA, MARC ROTHMANN, MARCO TASSEMEIER, THOMAS WIEMANN, and MARIO PORRMANN,
Osnabrück University, Germany

In this paper, we present an approach to integrate reconfigurable SoCs into the well known Robot Operating System (ROS). Our method allows to implement hardware-accelerated algorithms on FPGA and integrate them directly into the ROS ecosystem. This allows to combine the established and well tested ROS infrastructure together with low-power hardware acceleration. As a proof-of-concept for this novel integration, we ported an existing path-following algorithm onto an FPGA and tested it on an unmanned ground vehicle (UGV).

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

Additional Key Words and Phrases: ROS, FPGA, SoC, UGV, UAV, path detection

ACM Reference Format:
Marc Eisoldt, Marcel Flottmann, Julian Gaal, Steffen Hinderink, Juri Vana, Marc Rothmann, Marco Tassemeier, Thomas Wiemann, and Mario Porrmann. 2018. ReconfROS: Running ROS on Reconfigurable SoCs. In *DroneSE-2021: Drone Systems Engineering, January 18, 2021,Budapest, Hungary*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

The Robot Operating System (ROS) [20] is a popular robotic middleware used in research, education and industry. Due to the large community behind ROS, many now standard algorithms for basic tasks like localization, control and mapping have become freely available within this framework, making it the de-facto standard in the context of UGV and UAV development. The ROS system itself is designed to run on classical CPU/GPU systems. However, recent development has shown that FPGAs can outperform CPUs and GPUs in terms of power-efficiency for a wide range of applications [16]. Fixed-function ASICs are the most energy efficient alternative to off-the-shelf CPUs and GPUs. Their main drawbacks are high manufacturing cost and low flexibility as custom designs, since architectural changes are impossible after fabrication. FPGAs, in contrast, are flexible after fabrication, while offering high power efficiency and strong performance benefits for algorithms that are parallelizable. Also, many FPGAs offer partial dynamic reconfiguration, enabling time multiplexing of different hardware kernels at runtime. Since not all parts of an application can be computed efficiently on FPGAs, System-on-Chip (SoC) architectures combining CPU cores and FPGA-fabrics on a single chip are available. With a tight low-latency interconnect between the processors and the reconfigurable logic, these platforms combine the benefits of both worlds, making them ideal candidates for utilization in robot vehicle platforms. To effectively make use of such FPGA-based hardware accelerators in ROS applications, they need to be integrated into the ROS system. Besides hardware acceleration, they are also apt candidates for reduction of power consumption, which is an important limitation factor for the deployment of unmanned vehicle platforms like

UGVs and UAVs, where power consumption has a direct impact on the operating time due to the limited energy budget. With robots performing in increasingly demanding applications, the applied sensor systems used are also becoming more and more complex, resulting in a steadily growing need for computational power while meeting tight energy limits [24].

In this paper, we propose an approach to integrate reconfigurable SoCs with FPGAs directly into ROS, allowing to combine the benefits of high parallelism and low power consumption with the already existing drivers, controllers and algorithms available in ROS. To demonstrate the advantages of such a heterogeneous integration, we ported the path-following method presented in [1] to an FPGA-accelerated version integrated in the UGV Volksbot [26] platform using ROS. Here, we use the FPGA implementation to greatly reduce power consumption and processing time compared to the original implementation. The integration in ROS allows us to directly use the on-board cameras using the supplied drivers and camera messages. The steering commands generated by the FPGA-implementation are used directly to control the UGV via the existing ROS interfaces. This transparency allows us to easily compare the results of the hardware-accelerated implementation with the software reference. In the evaluation, we show that the proposed system layout can be used to speed up computation while significantly reducing the overall power consumption.

2 RELATED WORK

The combination of FPGAs and reconfigurable SoCs have become more and more popular on robot vehicle platforms in recent years. Many implementations use FPGAs and SoCs for image processing, taking advantage of the high parallelizability of these algorithms [18]. He et al. [8] used an SoC to implement an algorithm to allow an UAV to return to its base station using a down facing camera. Shene et al. [18] stabilized the video input on a mobile robot for improved image processing. For such applications, Xilinx provides a Vision Library [28], offering simple image filters and more sophisticated algorithms like stereo matching. It can be used for fast and efficient development of FPGA-accelerated image processing pipelines. Another family of algorithms that can be accelerated with reconfigurable devices are neural networks. Several works use neural networks on FPGAs [23, 25, 27, 31] for robotic purposes. If the computing requirements exceed the capabilities of reconfigurable SoCs, combinations of discrete CPUs and FPGAs are used. In [19], a complete control system for robots is presented, which accelerates crucial algorithms like Simultaneous Localization and Mapping (SLAM), motion planning and neural networks concurrently on a dedicated FPGA. Further examples of utilizing FPGAs in robotics are implementations of efficient Kalman Filters for self-localization of robot vehicle platforms [4, 5], SLAM [2, 22], and fuzzy controllers [17].

Several publications use FPGAs in ROS. Cheng et al. [3] implemented a ROS node on the processing system of an SoC, which communicates with an accelerator for the neural network YOLOv2 on the programmable logic. In a contribution to the FPT Design Contest, Nitta et al.[10] built a robot with an FPGA SoC as central processing device. The FPGA is used for image processing, while ROS nodes running on the ARM core are used for typical robotics tasks like line tracing and obstacle detection. In a subsequent publication, traffic signal detection was implemented on the FPGA as well [11]. A similar robot was built by Hasegawa et al. [7], using the FPGA to capture images from a camera and to generate PWM signals for a motor driver while running ROS on the processing system of the SoC. Moréac et al. [9] proposed a framework for the development of HW/SW embedded systems, including ROS and Gazebo-based hardware-in-the-loop simulation, and applied it to vision-based emergency landing in UAVs. The reconfigurable fabric contains accelerators for either tracking, detection, or emergency landing. The system can switch between these accelerators via dynamic partial reconfiguration. The processing system of the SoC executes two ROS nodes: a sensor interface node and the mission manager. For simulation, the SoC uses Ethernet to communicate with a host PC running the Gazebo simulator.

During the operation of the real UAV, the SoC sends information such as waypoints to a PixHawk board, which controls the UAV. In addition to handling the communication with the drone or simulation, the mission manager node also controls the hardware accelerators.

While the publications mentioned above use FPGAs in ROS systems, the integration of FPGAs into the ROS system is not their main contribution. Yamashina et al. [30] proposed a component to integrate FPGAs into ROS systems by encapsulating the FPGA and connecting it to ROS nodes to handle communication. The ROS Nodes are running on the processing system of an SoC. One ROS node is used as a subscriber and another ROS node as a publisher, while both ROS nodes communicate to the programmable logic of the SoC. To build upon their work, an automated design tool called cReComp was implemented, which automatically generates interface-circuitry and software for FPGA-based user logic [29]. Ohkawa et al. [14] proposed a system for architecture exploration using the ROS-compliant component and applied it to an implementation of VSLAM. Subsequently, the same system was applied to image processing and sensor fusion [13]. Sugata et al. [21] compared the communication latency of PC/PC communication and PC/SoC communication, concluding that the latency is higher for the SoC. They reduced the communication overhead by implementing the TCP/IP portion of the ROS protocol in hardware. Furthermore, Ohkawa et al. [12] explored the use of high-level synthesis to implement ROS communication in hardware. A different methodology to augment ROS with FPGA designs was suggested by Podlubne et al. [15].

The papers summarized previously mostly relied on a Linux distribution running on the processing system of an SoC to handle the ROS communication, while only *specific* tasks were implemented with programmable logic. Using the approach proposed in [15], everything necessary for communication with the ROS system can be implemented in hardware, and no SoC with a processing system running a Linux distribution is necessary. ROS publishers and subscribers are implemented in hardware. A TCP/IP interface on the FPGA controls an external network module, which includes a TCP/IP stack. A protocol generator handles the decoding and encoding of data for ROS messages on the FPGA.

While the presented implementations mostly concentrate on the development of dedicated and very specialized interface nodes with integrated preprocessing, in this paper we focus on general algorithmic acceleration. Our main contribution is to provide means for easy integration of any FPGA-based hardware accelerators into ROS nodes for more general *algorithmic* purposes other than stream processing or simple provision of dedicated specialized ROS nodes or system variants.

3 HARDWARE-SOFTWARE PLATFORM

3.1 Platform

In order to rapidly develop drones or other autonomous systems with ROS, a solid and easy to use base framework is needed for efficient integration of FPGAs. In this section, we propose ReconfROS, our approach for a hardware-software platform for ROS-based systems on a System-on-Chip. While the reference implementation is based on a Xilinx SoC, the approach is applicable to any SoC that provides a CPU compatible with Linux, shares (parts of) its memory with an FPGA and provides adequate connectivity for sensors and networking. In principle, the embedded processor is used as the basis for the ROS installation, while the algorithmic calculations are accelerated on the FPGA. This enables efficient solving of robotics tasks in power constrained environments and retains the flexibility and tooling of ROS, which is typically found on desktop class computers.

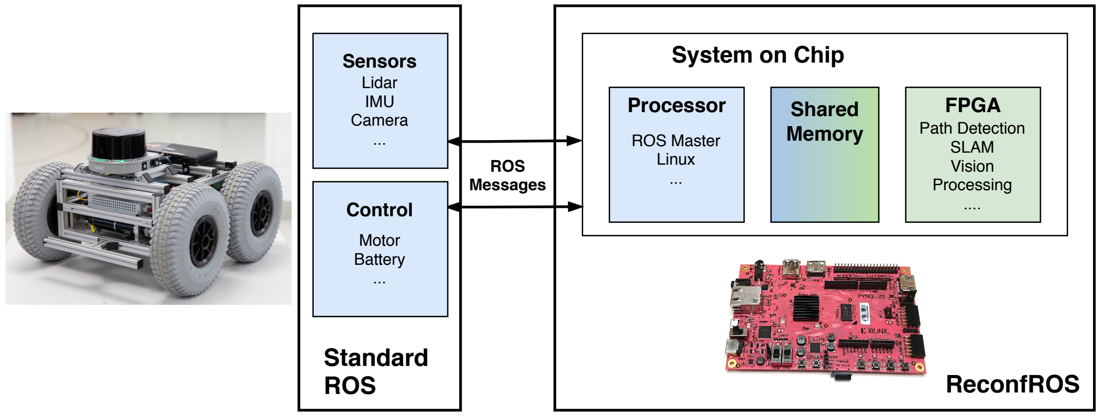


Fig. 1. The Ceres UGV (left) and an overview of the standalone version of the hardware software platform running ReconfROS.

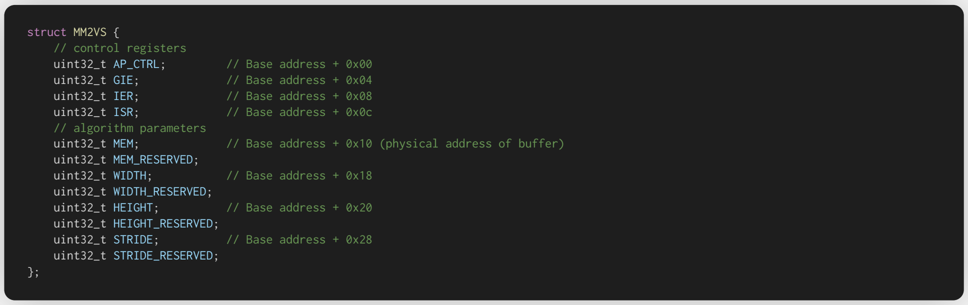
ROS provides standardized sensor drivers while algorithms accelerated on hardware give the opportunity to significantly reduce power draw and improve algorithmic performance. The dynamic reconfiguration and function loading of the FPGA can be used to implement different steps of an algorithm time-multiplexed on the same hardware to adapt to changing environmental conditions at runtime. This feature not only enhances flexibility, but also improves power efficiency further by keeping static power loss at a minimum without the need of a more power hungry, more expensive FPGA.

In ReconfROS, two modes of operation are possible: The SoC can be either used with its own roscore and connected sensors (see Fig. 1), or it can be interfaced as an external node connected via Ethernet to a larger ROS system, providing great flexibility and upgrade potential for existing robotic projects. For instance, our reference implementation is added to an existing mobile robot as an additional component that provides path information from a camera. It is also possible to run the pre-existing nodes on our platform and connect all sensors and actuators to it.

3.2 FPGA accelerated ROS Node

The FPGA Node on the CPU is responsible for controlling the FPGA. On startup, it initializes the interfaces necessary for communication with processing blocks in the hardware. Commonly, the registers of the processing blocks in the FPGA are mapped to the node's virtual memory. This way, the node can set the parameters prior to running the processing itself. The same interface controls starting and stopping the processing blocks as well. In complex algorithms that work with large inputs or outputs, e.g., image processing, using only registers to transfer data between FPGA and CPU is a major bottleneck. Therefore, the FPGA itself loads data from the main memory through dedicated memory ports instead. The ROS node only sets algorithm parameters and memory addresses via registers, while all other data is written to or read from exactly these memory addresses in main memory.

Ordinarily allocated memory is not necessarily contiguous, because every process has its own virtual memory space. The Contiguous Memory Allocator (CMA) in the Linux kernel provides functions for device drivers to allocate contiguous memory. A kernel module makes the CMA available for user space applications, e.g., the Xilinx Accelerator driver. It allocates contiguous regions called buffers of the required size and can be mapped to the virtual memory space of the requesting process. It also provides functions to flush and invalidate the cache of the CPU, as the FPGA cannot access main memory through the CPU cache. The register map of the processing blocks is represented as a plain data



```
struct MM2VS {
    // control registers
    uint32_t AP_CTRL;         // Base address + 0x00
    uint32_t GIE;             // Base address + 0x04
    uint32_t IER;             // Base address + 0x08
    uint32_t ISR;             // Base address + 0x0c
    // algorithm parameters
    uint32_t MEM;             // Base address + 0x10 (physical address of buffer)
    uint32_t MEM_RESERVED;
    uint32_t WIDTH;           // Base address + 0x18
    uint32_t WIDTH_RESERVED;
    uint32_t HEIGHT;          // Base address + 0x20
    uint32_t HEIGHT_RESERVED;
    uint32_t STRIDE;          // Base address + 0x28
    uint32_t STRIDE_RESERVED;
};
```

Fig. 2. An example of a register map data structure for a processing block that reads an image from memory and converts it to an FPGA internal stream.

structure to provide more readable code and type safety. To access the registers, the physical address range has to be mapped to the virtual memory space of the node. This is achieved by using the *mmap* system call and the */dev/mem* character device and casting the returning pointer to the appropriate data structure as shown in Fig. 2.

The buffers created with the CMA allocator consist of two addresses: a physical and a virtual. The node passes the physical address to the registers of the processing block in the FPGA and uses the virtual address to access the buffers for reading and writing data itself. Loading data into the FPGA works by writing data to the virtual address of the buffer and flushing the cache to make the FPGA aware of the changes to the address by the ROS node.

Regarding the ROS functionality, the node subscribes to and publishes topics as usual via Publishers and Subscribers. When data is received, the registers of the processing block are set accordingly and the buffers are allocated if needed and filled. It then starts the processing, waits for the FPGA to complete its calculations and publishes the calculated results on a ROS topic. This allows other nodes to use the FPGA to offload computationally expensive tasks.

For example, in our reference implementation the FPGA node subscribes to a camera topic and fills the buffer with images. After starting the FPGA calculations by setting the appropriate AP_CTRL signals (Fig. 2), the embedded CPU waits for the AP_DONE signal from the FPGA. Only then should the cache be invalidated by the appropriate CMA function to make the changes available to the CPU, after which the FPGA accelerated node can access the processed image at the virtual memory address. The actual output data of the algorithm is available through registers, since it only consists of a small set of values.

Due to the usage of the ROS infrastructure, an FPGA node can be replaced by a software prototype in early stages of the development to evaluate the functionality. When the prototype is implemented, the real hardware can be used in a hardware-in-the-loop scenario to verify the expected behavior: Simulation and visualization tools provided by ROS can be used for hardware verification during the process of adapting the prototype to hardware accelerated code.

4 REFERENCE IMPLEMENTATION

To demonstrate the performance and flexibility of ReconfROS, we implemented a reference algorithm in software and hardware. The goal is to accelerate an image-based trail detection algorithm on an FPGA and use the result for motor control of an UGV to autonomously follow a trail.

4.1 Hardware Setup

Our UGV, "Ceres", is based on the Volksbot platform [26] with ROS support for motor control [6]. It is equipped with a "PYNQ-Z2" FPGA board based on the Xilinx Zynq XC7Z020 SoC for hardware acceleration. The SoC consists of an ARM

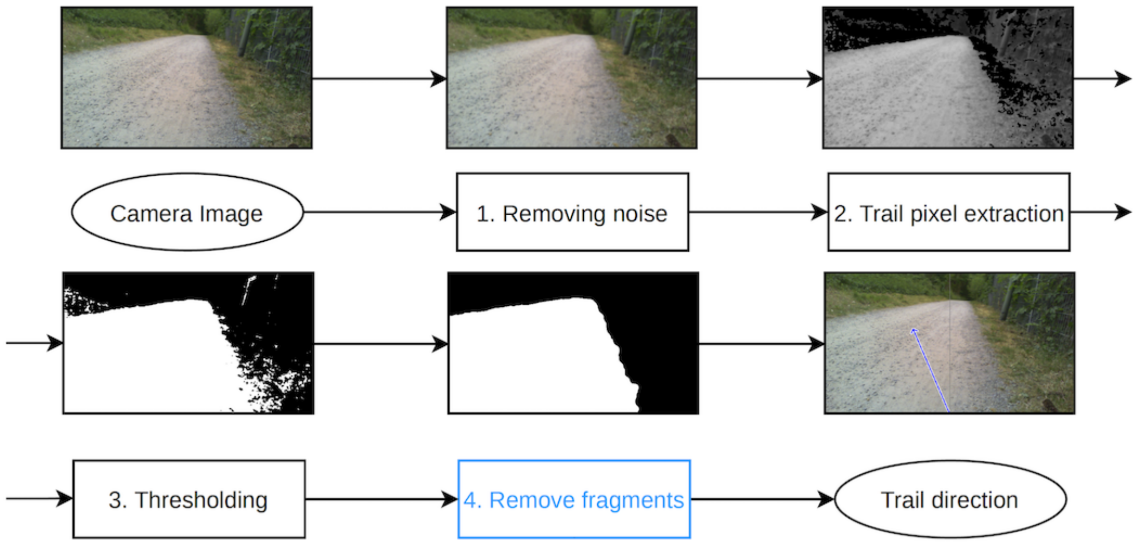


Fig. 3. Image processing pipeline. The steps in the black boxes correspond to the original implementation. The removal of fragments was replaced with morphological operations to allow efficient computation on the FPGA.

processor to launch the FPGA-accelerated ROS node and the communication between the FPGA and ROS. Additionally, an Intel NUC (NUC6i7KYK with Intel i7-6770HQ) is integrated for pure CPU-based reference implementations. This computer runs the ROS master and is therefore the central communication platform between all components in the system. For the algorithm, only a camera is needed, which was attached to the front of the robot. Since the UGV is intended to follow the trail autonomously, the motor driver is integrated into the ROS workspace and controlled by a dedicated navigation node that processes the results of either the FPGA-based trail detection algorithm, or the reference implementation in software.

4.2 Basic Algorithm

The trail detection algorithm is based on [1] and adapted as shown in Fig. 3. In the first step of the pipeline, Gaussian blur filtering is applied to reduce noise and produce more consistent results in the following parts of the algorithm. For path recognition, pixels that belong to this trail are transformed to white and others to black in the next two steps of the pipeline. The basic idea is to detect pixels that belong to the surrounding areas of the path, which mostly consist of green content. As shown in Fig. 3, the considered trail has a higher brightness compared with the surrounding environment after the green pixels have been removed. All pixels above a defined threshold are considered as part of the path and are marked white. These pixels are then used to extract the outer contour of the trail. At this point, the new algorithm differs from the original one (marked blue in Fig. 3). Typically, binary thresholding is effective, but leaves two types of fragments: holes inside of the path, and patches outside of the path that can significantly alter the calculated driving angle. To resolve this issue, in the original version of the algorithm, contours are extracted from the threshold image and only the largest is assumed to be the path.

Since this a complex operation not suitable for FPGA implementation, we remove the unwanted pixels using morphological operations. Holes are removed using the closing operation and small clusters outside of the path are

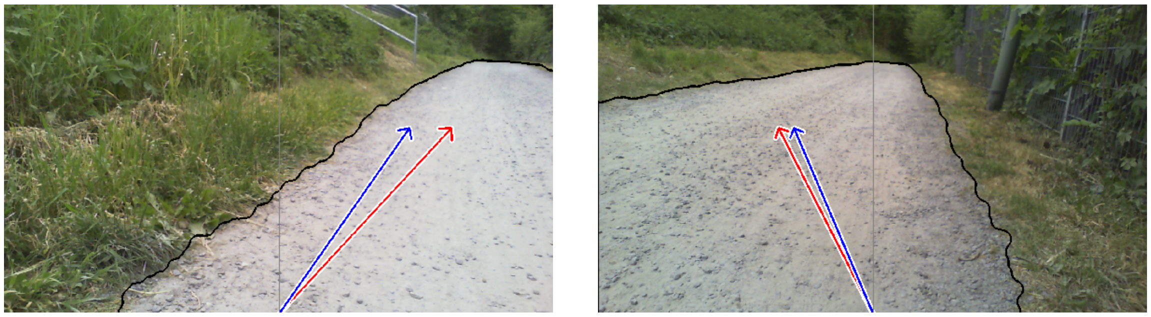


Fig. 4. Comparison of the results of the original algorithm (red) and the algorithm with our modifications (blue). The generated control signals differ slightly, but still deliver safe results to control the Ceres UGV.

thinned out by the opening operation. This also leads to a more uniform path contour. In the last step of the pipeline, the positions of the trail pixels are averaged on a fixed number of horizontal lines. These averages can be considered as ideal waypoints for the robot to follow. Based on the weighted sum of these points, the optimal driving angle can be derived. The direction of movement for the robot calculated this way is visualized blue arrow in the lower right image of Fig. 3. Additional examples are provided in a supplemental video [1] and the experimental results section.

4.3 FPGA Integration

For parameter-tuning, a pure software prototype was implemented and tested in a simulated environment. Using recorded data and the simulation software Gazebo in combination with ROS' dynamically reconfigurable parameters, the parameterization process was significantly sped compared to parameterization on the FPGA SoC directly. Thanks to the standardized interfaces of ROS, the different execution environments could be exchanged easily. The main advantage of using the simulation is that all main components can be tested, including the navigation node itself. Almost all steps of the software version of the trail detection algorithm are implemented using functions from OpenCV. An advantage of the introduced adaptation of the original algorithm in this paper is that all major steps can be implemented using high-level synthesis using the Xilinx HLS Video Library [28].Regarding the optimization of the hardware accelerator, each processing step from Fig. 3 was optimized following standard high-level synthesis techniques. Furthermore, the high-level synthesis was directed to fully pipeline these steps, resulting in a high-throughput implementation. Both the hardware accelerated and the software implementation were tested in Gazebo and ROS bagfiles with reference camera data to ensure that worked as expected for a real navigation task on Ceres.

5 EXPERIMENTAL RESULTS

To evaluate the ideas of ReconfROS, we analyze the path detection algorithm discussed in the previous section with respect to deviations from the original results due to our modifications, run time performance and power consumption. All experiments concerning power and performance were conducted using the modified version in of the algorithm to ensure comparability. For that, we also implemented a software version of the modified algorithm and verified that both versions produce the same results. The measurements of the software implementation were performed on the Intel NUC of the Ceres robot, while the FPGA-accelerated implementation was tested on the PYNQ-Z2 board.

[1]https://youtu.be/WSk_ug7hgkQ

Table 1. Available resources on the FPGA and used quantities of our hardware implementation.

Resource	Available	Utilized	Utilization [%]
LUT	53200	13750	25.8
LUTRAM	17400	854	4.9
FF	106400	15148	14.2
BRAM	140	35.5	25.4
DSP	220	78	35.5
BUFG	32	1	3.1

Table 2. Runtime and power consumption measurements of the algorithm in software and hardware.

	Runtime		Power Consumption		
	Average [ms]	Maximum [ms]	Idle [W]	Running [W]	Difference [W]
Software	52.352	73.573	18.5	21.8	3.3
Hardware	18.422	40.780	1.8	2.4	0.6

5.1 Comparison with the Baseline Algorithm

Two example frames from the recorded images used to control the Ceres robot are shown in Fig. 4. They show two situations where the UGV approaches the trail border and needs to adjust the steering direction. The arrows indicate the results by pointing to the calculated weighted average on the x-coordinate while fixing y, thus depicting the direction, in which the correction signal is produced. The direction signals generated by the original algorithm are drawn in red, those computed by the FPGA-adapted version are marked blue. In the first image, the robot is driving towards the left trail border, so the algorithm sends a steering command to the right. An inverse situation for the right trail border is shown in the right image. Although we replaced the fragment removal method, the generated steering signals do not significantly differ. In both implementations, the robot is able to reliably follow the path, although the original algorithm generally produced stronger turn signals resulting in “sharper” movements.

5.2 Usage of FPGA Resources

Tab. 1 shows how much of the available resources on the FPGA were utilized by the hardware implementation of the algorithm. The utilization shows that the FPGA on the PYNQ-Z2 offers sufficient headroom for further extensions or algorithmic improvements. Since ReconfROS allows for an easy exchange of the utilized SoC, a suitable device can be chosen, if more resources are needed to match the performance and resource requirements of the problem at hand.

5.3 Runtime

The runtimes of the implementation both in software and hardware were measured using one run of the same ROS bag file containing a raw image stream. The resolution was 1280×720 px. Tab. 2 shows the measured average and maximum runtime of the algorithm in software and hardware. The numbers indicate that, on average, the algorithm ran almost three times faster in hardware. Furthermore, the maximum runtime of one iteration of the algorithm in software was almost two times longer compared to hardware. Further testing showed that to run as fast as in hardware, the input image resolution needs to be less than 50% in both dimensions for the software implementation. So the prototypical hardware implementation allowed us to compute higher resolution images in shorter time in this proof-of-concept application.

5.4 Power Consumption

The second advantage of implementing algorithms directly in hardware is the significantly reduced power consumption. Tab. 2 shows the measured power consumption of the trail detection algorithm in software and in hardware For comparison, the power draw was measured in two scenarios for each system: idle and power draw with the algorithm activated. The reported difference between these values therefore reflects the additional power draw of the algorithm itself. In summary, using hardware-accelerated code, we were able to reduce the total consumed power by about 80%. More than 15 times less energy is needed to process one frame with the FPGA-accelerated ROS node. While the pure software implementation draws 0.172 J, the hardware implementation draws just 0.011 J of energy per frame.

6 CONCLUSION

We presented ReconfROS, a platform that combines the advantages of both ROS and reconfigurable SoCs. The easy management of different components of a robot system, for example various sensors, a navigation system or actuators, can be accomplished with ROS nodes, messages and topics. More importantly, computationally intensive, especially parallelizable algorithms in the system can be implemented in hardware. We demonstrated and tested our platform with a typical robotics use case. By using ReconfROS, the algorithm can be executed both significantly faster and more energy efficient.

Our approach is applicable to many more robotics algorithms and different reconfigurable SoCs matching specific algorithmic requirements. Ideally, in the scope of larger robotics problems, multiple algorithms could be accelerated in hardware at the same time, a method that has only been briefly discussed in this paper. To expand our ideas presented in this paper further, we are working on the implementation of a TSDF-based simultaneous localization and mapping (SLAM) challenge using ReconfROS methods on a more powerful SoC.

REFERENCES

[1] Andreas Bartel, Frank Meyer, Christopher Sinke, Thomas Wiemann, A Nchter, Kai Lingemann, and Joachim Hertzberg. 2007. Real-time outdoor trail detection on a mobile robot. In *Proceedings of the 13th IASTED International Conference on Robotics, Applications and Telematics*. 477–482.

[2] Konstantinos Boikos and Christos Savvas Bouganis. 2016. Semi-dense SLAM on an FPGA SoC. In *FPL 2016 - 26th International Conference on Field-Programmable Logic and Applications*. Institute of Electrical and Electronics Engineers Inc., 1–4. https://doi.org/10.1109/FPL.2016.7577365

[3] Haoxuan Cheng, Shimpei Sato, and Hiroki Nakahara. 2018. A Performance Per Power Efficient Object Detector on an FPGA for Robot Operating System (ROS). In *Proceedings of the 9th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies* (Toronto, ON, Canada) *(HEART 2018)*. Association for Computing Machinery, New York, NY, USA, Article 20, 4 pages. https://doi.org/10.1145/3241793.3241814

[4] Luis Contreras, Sergio Cruz, J. M.S.T. Motta, and Carlos H. Llanos. 2016. FPGA implementation of the EKF algorithm for localization in mobile robotics using a unified hardware module approach. In *2015 International Conference on ReConFigurable Computing and FPGAs, ReConFig 2015*. Institute of Electrical and Electronics Engineers Inc., 1–6. https://doi.org/10.1109/ReConFig.2015.7393315

[5] Sergio Cruz, Daniel M. Munoz, Milton Conde, Carlos H. Llanos, and Geovany A. Borges. 2013. FPGA implementation of a sequential Extended Kalman Filter algorithm applied to mobile robotics localization problem. In *2013 IEEE 4th Latin American Symposium on Circuits and Systems, LASCAS 2013 - Conference Proceedings*. Institute of Electrical and Electronics Engineers Inc., 1–4. https://doi.org/10.1109/LASCAS.2013.6519021

[6] Martin Günther, Sebastian Puetz, and Jochen Sprickerhof. 2013. ROS Volksbot Driver. https://github.com/uos/volksbot_driver. [Online; accessed 21-Nov-2020].

[7] K. Hasegawa, K. Takasaki, M. Nishizawa, R. Ishikawa, K. Kawamura, and N. Togawa. 2019. Implementation of a ROS-Based Autonomous Vehicle on an FPGA Board. In *International Conference on Field-Programmable Technology (ICFPT)*. IEEE, Tianjin, China, 457–460. https://doi.org/10.1109/ICFPT47387.2019.00092

[8] Qunfang He, · Wenjie Chen, Danping Zou, · Zhilei Chai, and Zhilei Chai. 2020. A novel framework for UAV returning based on FPGA. *The Journal of Supercomputing* (2020). https://doi.org/10.1007/s11227-020-03434-4

[9] Erwan Moréac, El Mehdi Abdali, François Berry, Dominique Heller, and Jean-Philippe Diguet. 2020. Hardware-in-the-loop simulation with dynamic partial FPGA reconfiguration applied to computer vision in ROS-based UAV. In *31st International Workshop on Rapid System Prototyping (RSP)*. Virtual Conference (ESWEEK), France. https://hal.archives-ouvertes.fr/hal-02948474

[10] Y. Nitta, S. Tamura, and H. Takase. 2018. A Study on Introducing FPGA to ROS Based Autonomous Driving System. In *2018 International Conference on Field-Programmable Technology (FPT)*. IEEE, Naha, Okinawa, Japan, 421–424. https://doi.org/10.1109/FPT.2018.00090

[11] Y. Nitta, S. Tamura, H. Yugen, and H. Takase. 2019. ZytleBot: FPGA Integrated Development Platform for ROS Based Autonomous Mobile Robot. In *International Conference on Field-Programmable Technology (ICFPT)*. IEEE, Barcelona, Spain, 445–448. https://doi.org/10.1109/ICFPT47387.2019.00089

[12] T. Ohkawa, Y. Sugata, H. Watanabe, N. Ogura, K. Ootsu, and T. Yokota. 2019. High Level Synthesis of ROS Protocol Interpretation and Communication Circuit for FPGA. In *2019 IEEE/ACM 2nd International Workshop on Robotics Software Engineering (RoSE)*. IEEE Computer Society, Los Alamitos, CA, USA, 33–36. https://doi.org/10.1109/RoSE.2019.00014

[13] Takeshi Ohkawa, Kazushi Yamashina, Hitomi KIMURA, Kanemitsu Ootsu, and Takashi YOKOTA. 2018. FPGA components for integrating FPGAs into robot systems. *IEICE Transactions on Information and Systems* E101.D (02 2018), 363–375. https://doi.org/10.1587/transinf.2017RCP0011

[14] Takeshi Ohkawa, Kazushi Yamashina, Takuya Matsumoto, Kanemitsu Ootsu, and Takashi Yokota. 2016. Architecture Exploration of Intelligent Robot System Using ROS-Compliant FPGA Component. In *Proceedings of the 27th International Symposium on Rapid System Prototyping: Shortening the Path from Specification to Prototype* (Pittsburgh, Pennsylvania) *(RSP '16)*. Association for Computing Machinery, New York, NY, USA, 72–78. https://doi.org/10.1145/2990299.2990312

[15] A. Podlubne and D. Göhringer. 2019. FPGA-ROS: Methodology to Augment the Robot Operating System with FPGA Designs. In *International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. IEEE, Cancun, Mexico, 1–5. https://doi.org/10.1109/ReConFig48160.2019.8994719

[16] Murad Qasaimeh, Joseph Zambreno, Phillip H. Jones, Kristof Denolf, Jack Lo, and Kees Vissers. 2019. Analyzing the energy-efficiency of vision kernels on embedded cpu, GPU and FPGA platforms. In *Proceedings - 27th IEEE International Symposium on Field-Programmable Custom Computing Machines, FCCM 2019*. Institute of Electrical and Electronics Engineers Inc., 336. https://doi.org/10.1109/FCCM.2019.00077

[17] Santiago Sánchez-Solano, Alejandro J. Cabrera, Iluminada Baturone, Francisco J. Moreno-Velo, and María Brox. 2007. FPGA implementation of embedded fuzzy controllers for robotic applications. *IEEE Transactions on Industrial Electronics* 54, 4 (aug 2007), 1937–1945. https://doi.org/10.1109/TIE.2007.898292

[18] Tahiyah Nou Shene, K. Sridharan, and N. Sudha. 2016. Real-Time SURF-Based Video Stabilization System for an FPGA-Driven Mobile Robot. *IEEE Transactions on Industrial Electronics* 63, 8 (aug 2016), 5012–5021. https://doi.org/10.1109/TIE.2016.2551684

[19] Xuesong Shi, Lu Cao, Dawei Wang, Ling Liu, Ganmei You, Shuang Liu, and Chunjie Wang. 2018. HERO: Accelerating Autonomous Robotic Tasks with FPGA. In *IEEE International Conference on Intelligent Robots and Systems*. Institute of Electrical and Electronics Engineers Inc., 7766–7772. https://doi.org/10.1109/IROS.2018.8593522

[20] Stanford Artificial Intelligence Laboratory et al. [n.d.]. *Robotic Operating System*. https://www.ros.org

[21] Yuhei Sugata, Takeshi Ohkawa, Kanemitsu Ootsu, and Takashi Yokota. 2017. Acceleration of Publish/Subscribe Messaging in ROS-Compliant FPGA Component. In *Proceedings of the 8th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies* (Bochum, Germany) *(HEART2017)*. Association for Computing Machinery, New York, NY, USA, Article 13, 6 pages. https://doi.org/10.1145/3120895.3120904

[22] Keisuke Sugiura and Hiroki Matsutani. 2020. An FPGA Acceleration and Optimization Techniques for 2D LiDAR SLAM Algorithm. arXiv:2006.01050 [eess.SP]

[23] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Vol. 07-12-June-2015. IEEE Computer Society, 1–9. https://doi.org/10.1109/CVPR.2015.7298594 arXiv:1409.4842

[24] Zishen Wan, Bo Yu, Thomas Yuang Li, Jie Tang, Yuhao Zhu, Yu Wang, Arijit Raychowdhury, and Shaoshan Liu. 2020. A Survey of FPGA-Based Robotic Computing. arXiv:2009.06034 [cs.RO]

[25] Ying Wang, Jie Xu, Yinhe Han, Huawei Li, and Xiaowei Li. 2016. DeepBurning: Automatic Generation of FPGA-Based Learning Accelerators for the Neural Network Family. In *Proceedings of the 53rd Annual Design Automation Conference* (Austin, Texas) *(DAC '16)*. Association for Computing Machinery, New York, NY, USA, Article 110, 6 pages. https://doi.org/10.1145/2897937.2898003

[26] Thomas Wisspeintner, Walter Nowak, and Ansgar Bredenfeld. 2006. VolksBot – A Flexible Component-Based Mobile Robot System. In *RoboCup 2005: Robot Soccer World Cup IX*, Ansgar Bredenfeld, Adam Jacoff, Itsuki Noda, and Yasutake Takahashi (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 716–723.

[27] Xilinx. 2020. Vitis AI. https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html Accessed 2020-11-02.

[28] Xilinx. 2020. Vitis Vision Library. https://xilinx.github.io/Vitis{_}Libraries/vision/2020.1/index.html Accessed on 2020-11-02.

[29] K. Yamashina, H. Kimura, T. Ohkawa, K. Ootsu, and T. Yokota. 2016. cReComp: Automated Design Tool for ROS-Compliant FPGA Component. In *2016 IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSOC)*. 138–145. https://doi.org/10.1109/MCSoC.2016.47

[30] Kazushi Yamashina, Takeshi Ohkawa, Kanemitsu Ootsu, and Takashi Yokota. 2015. Proposal of ROS-compliant FPGA Component for Low-Power Robotic Systems. arXiv:1508.07123 [cs.AR]

[31] Xiaofan Zhang, Haoming Lu, Cong Hao, Jiachen Li, Bowen Cheng, Yuhong Li, Kyle Rupnow, Jinjun Xiong, Thomas Huang, Honghui Shi, Wen mei Hwu, and Deming Chen. 2020. SkyNet: a Hardware-Efficient Method for Object Detection and Tracking on Embedded Systems. arXiv:1909.09709 [cs.CV]