Thundershot: an open-source autonomous vehicles research platform for embedded heterogeneous MPSoCs

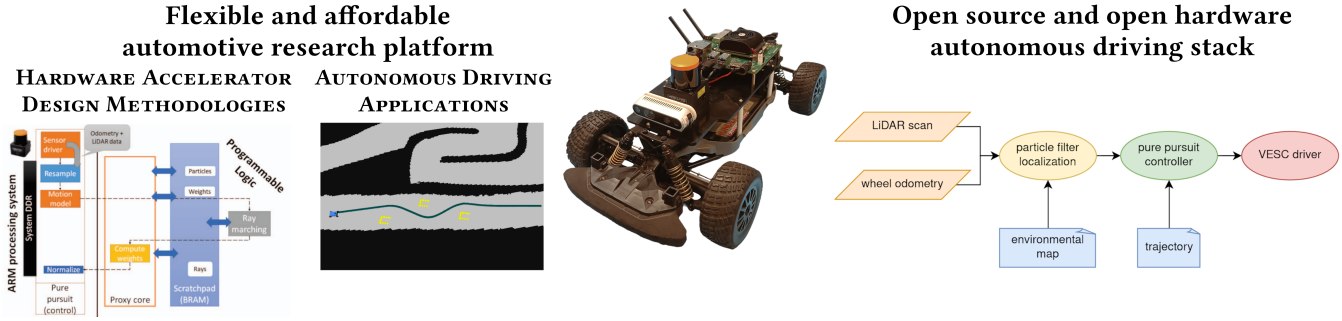
Federico Gavioli
Università di Modena e Reggio Emilia
Modena, Italy
federico.gavioli@unimore.it

Francesco Moretti
Università di Modena e Reggio Emilia
Modena, Italy
francesco.moretti@unimore.it

Antonio Russo
Università di Modena e Reggio Emilia
Modena, Italy
270201@studenti.unimore.it

Alessandro Capotondi
Università di Modena e Reggio Emilia
Modena, Italy
alessandro.capotondi@unimore.it

Paolo Burgio
Università di Modena e Reggio Emilia
Modena, Italy
paolo.burgio@unimore.it



Abstract

Modern autonomous vehicles require efficient and predictable hardware and software to guarantee a high level of safety. Meeting response time deadlines while processing large amounts of sensor data and maintaining a reasonable power consumption is a complex task on embedded devices. Hardware acceleration on FPGA fabric proposes itself as a predictable and certifiable solution to this problem. Research on embedded heterogeneous platforms, however, has a more complex development lifecycle. This paper presents a complete hardware/software platform for autonomous driving research on a commercial off-the-shelf Industrial-Grade FPGA-based MPSoC (an AMD KR260), with a particular focus on autonomous racing use-cases. On the software side, we release the first iteration of our open-source autonomous racing stack based on ROS2. Finally, we present a case-study on a hardware-accelerated 2D LiDAR localization pipeline, which is developed and integrated on the platform. The FPGA implementation provides a 2.64x speed-up over its host counterpart and is released as well as open hardware.

CF Companion '25, Cagliari, Italy

ACM ISBN 979-8-4007-1393-4/25/05
https://doi.org/10.1145/3706594.3726968

CCS Concepts

• **Hardware** → **Hardware accelerators**; **Reconfigurable logic applications**; • **Computer systems organization** → **Embedded systems**; **Embedded software**; **Embedded hardware**; **Robotics**.

Keywords

Hardware accelerators, Embedded systems, Robotics

ACM Reference Format:
Federico Gavioli, Francesco Moretti, Antonio Russo, Alessandro Capotondi, and Paolo Burgio. 2025. Thundershot: an open-source autonomous vehicles research platform for embedded heterogeneous MPSoCs. In *22nd ACM International Conference on Computing Frontiers (CF Companion '25), May 28–30, 2025, Cagliari, Italy*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3706594.3726968

1 Introduction

Over the past few decades, autonomous driving has been an active research topic both in the industrial and academic world. Be it urban vehicles, racing prototypes or autonomous delivery robots, research on full-scale vehicles like [7] adds a lot of costs and complexity, while also requiring dedicated testing environments. To avoid incurring these hidden costs and depending on the requirements, the solutions to this problem include the use of simulators or scaled prototypes. The latter is usually a cost-effective alternative that can accurately mimic the hardware architecture and sensor set of a full-scale vehicle, but without most of their issues.

Industrial-grade autonomous driving software stacks like Autoware [8] are very complex systems composed of many different

modules, requiring a high amount of expertise to master all of the topics faced. The availability of a research-oriented and easy to use open-source software stack is crucial to provide a solid knowledge base to develop and test new approaches.

Automotive software usually requires predictable platforms to provide safety and reliability guarantees. Although FPGA-based platforms offer a higher energy-efficiency and a much more inspectable environment, hardware design has a long and complex development cycle when compared with traditional software lifecycles. While High-Level Synthesis (HLS) is usually an acceptable solution to this problem and some industrial [11] and academic [6] works address these issues, the integration between hardware accelerators and traditional CPUs is a challenge yet to be completely solved in matters of performance, flexibility and ease of use.

In this work, we first present a completely open-source hardware/software autonomous vehicles research platform based on industrial-grade commercial off-the-shelf components. We base our prototype on an AMD Kria KR260.

Secondly, we release as the first iteration of our autonomous racing stack as open-source software[1]. This implementation is developed and tested specifically for the proposed platform and contains the basic nodes to localize and navigate the vehicle across a path.

Finally, we present a case study on the design and implementation of a hardware accelerator for vehicle localization. The design is validated on the an AMD KR260 and integrated with the software stack. The hardware accelerator and the system design are released as open hardware[2].

2 Related works and target platform

An important project in the context of small-scale vehicles is Roboracer [12], previously known as F1tenth. This project was born to research autonomous driving in racing contexts and is backed by a large community that frequently organizes races at academic conferences. The cars used in the Roboracer competitions use a 2D LIDAR, a camera, and an IMU as their main data sources. Roboracer vehicles are based on embedded Nvidia Jetson computers. While these platforms can be used to conduct research on embedded systems and GPU-related topics, there is no official version based on a FPGA System on Chip (SoC).

There are also some examples of platforms that already use a SoC FPGA as the main computing board. Kojima [9] proposes a small-scale FPGA-based prototype for urban scenarios. While this work demonstrates the advantages that the integration of FPGA accelerators can bring to the field of autonomous driving, it is important to note that the software stack for the project is not open-source, thus it is difficult to replicate and extend.

In addition to being open-source, another important aspect for a platform to have a significant impact is its compatibility with existing solutions in the field. In autonomous driving, as well as in robotics in general, the typical development process involves the use of a publish/subscribe framework to split the software into different components. The most used framework in this field is ROS2[10]. This approach offers multiple advantages, including modularity, increased efficiency in development, and interoperability of the software components. Especially the latter can be a key aspect for research purposes, as it facilitates the integration of the case study with the chosen platform.

The authors of HydraMini [14] implemented an extensible FPGA-based platform, but implemented their own message-passing solution for node communication. For this reason, integrating existing software modules is not as easy as it is in a system based on ROS2.

Some examples of a complete and extendable software stack can be found in the Roboracer project cited previously. In addition to the official stack that the organizers have made available, some participants also released their autonomous driving stack or a portion of it, making the whole community progress. An example of this is the ForzaETH Race Stack [2], an autonomous racing stack released by one of the most active teams in the competition. Still, none of the Roboracer prototypes provides any FPGA acceleration capabilities.

Our system took inspiration from the previous works, trying to combine their advantages into a single open and modular solution. Our work not only supports FPGA acceleration, but is also easily extendable thanks to the integration with the ROS2 ecosystem.

3 Open-source platform and racing stack

The proposed prototype is inspired by the Roboracer research platform. Its main sensor is a 2D LiDAR rangefinder, namely a Hokuyo UST-10LX. Other available sensors on the vehicle include a BMI160 inertial sensor and a Realsense D435i stereo camera. A wheel odometry can also be derived by the feedback data of the on-board VESC (Vedder Electric Speed Controller), which is also used to control both the main motor and the steering servomotor. While this is the basic provided configuration, the platform is easily extensible to other sensors (e.g. multi-line 3D LiDARs, Radars, etc.).

The main difference with the Roboracer platform is the central computing board. The proposed prototype is developed over an AMD KR260, a low-cost commercial off-the-shelf device targeting the embedded computer vision and robotics market. It hosts a Zynq Ultrascale+ MPSoC with a Quad-core ARM Cortex A53 @1.3 GHz on the processing system side coupled with an FPGA accelerator as general-purpose programmable logic.

Since the use-case for the vehicle is autonomous racing, the software aims to steer the vehicle over a provided reference path in the fastest way possible.

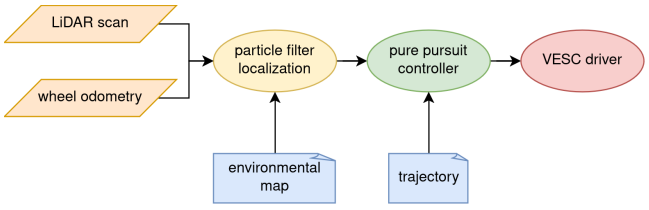


Figure 1: The proposed autonomous racing stack.

The proposed embedded racing stack is built on top of ROS2 and is mainly composed of two components: a particle filter for vehicle localization and a pure pursuit controller. Additional tools like race trajectory optimization tools and mapping solutions are omitted as they are usually run on dedicated hardware outside of the vehicle, thus they're out of the focus of this paper. The racing stack mainly

[1] https://github.com/HiPeRT/unimore_roboracer/
[2] https://github.com/HiPeRT/particle_filter_hw/

leverages LiDAR scans and the vehicle's odometry to localize itself into a pre-mapped racetrack. The localization is subsequently used by a pure-pursuit controller to steer the vehicle over a racing line, which is generated and optimized off-board. The computation flow of the racing stack is summarized graphically in Figure 1.

Particle Filter: In order to provide an accurate localization, we use a particle filter [4]: a robust and widely used solution in robotics state estimation. In our implementation the pipeline is mainly composed of three steps: first, in the motion model step, the filter updates the particle poses using the wheel odometry data with added Gaussian noise to model inaccuracies. Secondly, a sensor model step is tasked with integrating the LiDAR measurement. This step generates a virtual scan for each particle based on its position and orientation on the map with the ray marching algorithm. This virtual scan is compared against the actual sensor measurement in order to assign a weight value to each particle. The final output state of the filter (and thus, of the vehicle) is computed as the weighted average of the position and orientation of the particle set. The ray marching is the most computational heavy section of the whole pipeline.

Pure Pursuit Controller: The pure pursuit algorithm [13] is a geometric control algorithm widely used in autonomous driving scenarios, allowing vehicles to follow a predetermined reference trajectory. This algorithm operates by determining a look-ahead point projected ahead on the path on the path. The vehicle then calculates the steering angle required to direct itself towards the look-ahead point, effectively steering along a circular arc that intersects both its current position and the look-ahead point.

The racing stack, along with support libraries and utilities, is released open-source on GitHub.

4 Accelerated particle filter for vehicle localization

Since vehicle localization is the most computational-heavy component of the racing stack, improving its execution latency is crucial for a smooth and responsive vehicle control. By profiling the particle filter implementation, we identified the ray marching as the most time-consuming section, around 70% of the full pipeline. This makes it the ideal target for hardware acceleration. In this section we design and implement a Ray Marching hardware accelerator and its integration into the previously detailed localization pipeline.

Accelerator design: The accelerator is designed using Vitis HLS based on the design of Bernardi et al. [3]. It implements the ray marching step of the particle filter. The Ray Marching (RM) engine is configured via an AXI4-Lite port with pointers to the DRAM input-output shared memory buffers, which are used to exchange data with the host processors. Each RM engine is made up of three main components: first, a data collector accesses the shared buffers and pulls the necessary data (the distance map and point cloud metadata) into a shared block RAM memory for faster access. This memory is array-partitioned in order to provide high-performance parallel access. Secondly, a high-performance engine computes the virtual ray distance for each of the supplied particles. This result is finally written back to the DRAM.

System design: The accelerator is deployed on the KR260 as a single engine with 32 processing elements running at 266.66 *MHz*.

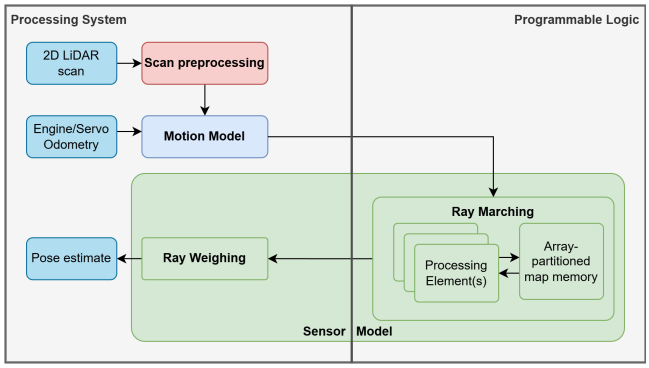


Figure 2: The proposed particle filter application design.

Since the device is particularly constrained in terms of Block RAM availability, we split the array-partitioned PL-side memory into Block RAM and Ultra RAM banks. With this configuration, the design requires 21.30% of the total LUTs and occupies respectively the 10.76% and 75.00% of the available BRAM and URAM.

Host-accelerator integration: In order to communicate with the hardware accelerator, the host code running inside the Linux user-space needs to access the accelerator registers provided on its AXI4-Lite configuration port. Then, we need to provide the accelerator with physical contiguous DRAM memory to fetch particle inputs and write back ray outputs. For this, we leverage the Linux CMA (Contiguous Memory Allocator) to allocate continuous reserved-memory buffers at the operating system level. Both the accelerator registers and the reserved buffers are exposed as UIO (Userspace I/O) devices instantiated on a custom hot-pluggable device-tree overlay. A simple driver is finally developed as an interface to the hardware accelerators, abstracting all the previously discussed mechanisms behind simple library calls.

Deployment flow: In order to easily deploy the ray marching accelerator, the system design bitstream is packaged together with its device-tree overlay into an xmutil application. The creation of such a package does not require building any custom kernel or additional modules and can be instantiated directly using the readily-available AMD Ubuntu images. Since everything is handled via user-space primitives, the accelerator driver is distributed as a C++ library.

The accelerator code and its system design are released as open-source on GitHub, while the driver is distributed along with the particle filter host code into the main autonomous stack repository.

5 Experimental results

This section validates the proposed hardware accelerator system against its original host implementation.

Since we are only profiling the localization pipeline and not the whole autonomous stack, we can rely on a Hardware-in-the-Loop (HiL) test infrastructure. First, we record sensor data and ground truth positions to build a validation dataset using the well-known F1tenth Gym simulator [1]. This dataset is then used for all the subsequent tests to validate the proposed system. A desktop PC is connected via a point-to-point Ethernet link to the KR260, providing sensor data to the vehicle and collecting the localization results.

The test runs are then compared against the simulator ground truth with evo [5], a popular set of evaluation tools for localization and mapping applications.

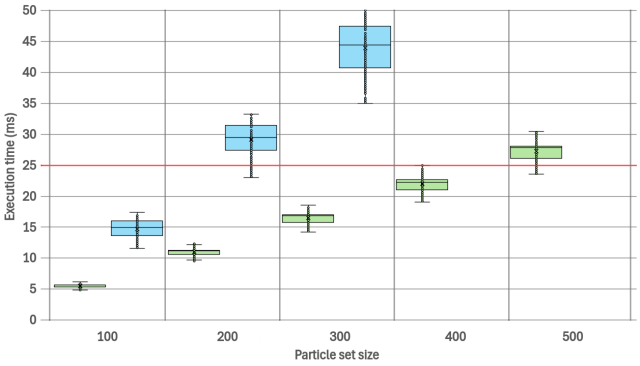


Figure 3: Average latency for increasing particle set sizes. Blue boxes represent latency data from the host implementation, while green boxes represent latency data from the FPGA-accelerated implementation. Results for 400+ particles running on the host processors are omitted due to filter divergence caused by high latency.

Figure 3 shows the average latency data over a single lap. First, we can notice the accelerated implementation performs significantly better than the host counterpart. We measured a speed-up of 2.64x with respect to the pure CPU implementation. The red line in the figure represents the maximum acceptable execution time of $25ms$, the time period after which a new LiDAR scan is available to be processed. Considering the case with 300 particles, which is a commonly acceptable solution for our localization use-case, the host implementation widely misses the deadline. Deadline misses lead to packet loss (since not all frames are processed) and a higher response time, hindering the whole control loop performance. On the other hand, the accelerated implementation performance lies fully under the required deadline even on the 400 particles configuration. Moreover, comparing similar execution times (e.g. the accelerated implementation at 300 particles and the host implementation at 200 particles), the variability in latency is much narrower in the FPGA implementation. This highlights and validates once more the higher predictability of the platform.

We also compare the localization accuracy of the method against its CPU counterpart. Despite the accelerator working on fixed-point values for efficiency reasons, it performs comparably to the CPU implementation, with a Root Mean Square Error (RMSE) of $7.2cm$ on a full lap in the highest comparable particle set size. This value is within the average error of the particle filter implementation, so we can conclude that no accuracy loss can be perceived in the accelerated application.

6 Conclusions and future works

This paper presents a complete and open-source platform based on embedded FPGAs. The proposed prototype is based on commercial hardware (an AMD Kria KR260) and can mount a wide variety of sensors. Along with the vehicle we also present the first iteration of our open-source software stack built on top of the ROS2 framework. To validate and demonstrate the usage of our platform, we finally proposed an hardware-accelerated implementation of our particle filter-based 2D LiDAR localization, which is fully integrated in the host-side autonomous driving stack. Our hardware-accelerated localization achieves an average 2.64x increase in latency without any perceivable difference on the resulting localization accuracy. The hardware accelerator is also released as open-source.

Acknowledgments

This work was supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001 - RESTART), project MoVeOver.

References

[1] [n. d.]. The F1tenth Gym simulator. https://github.com/f1tenth/f1tenth_gym.
[2] Nicolas Baumann, Edoardo Ghignone, Jonas Kühne, Niklas Bastuck, Jonathan Becker, Nadine Imholz, Tobias Kränzlin, Tian Yi Lim, Michael Lötscher, Luca Schwarzenbach, et al. 2024. ForzaETH Race Stack—Scaled Autonomous Head-to-Head Racing on Fully Commercial Off-the-Shelf Hardware. *Journal of Field Robotics* (2024).
[3] Andrea Bernardi, Gianluca Brilli, Alessandro Capotondi, Andrea Marongiu, and Paolo Burgio. 2022. An FPGA Overlay for Efficient Real-Time Localization in 1/10th Scale Autonomous Vehicles. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 915–920. doi:10.23919/DATE54114.2022.9774517
[4] Dieter Fox, Sebastian Thrun, Wolfram Burgard, and Frank Dellaert. 2001. *Particle Filters for Mobile Robot Localization*. Springer New York, New York, NY, 401–428. doi:10.1007/978-1-4757-3437-9_19
[5] Michael Grupp. 2017. evo: Python package for the evaluation of odometry and SLAM. https://github.com/MichaelGrupp/evo.
[6] Carsten Heinz, Jaco Hofmann, Jens Korinth, Lukas Sommer, Lukas Weber, and Andreas Koch. 2021. The TaPaSCo Open-Source Toolflow: for the Automated Composition of Task-Based Parallel Reconfigurable Computing Systems. *J. Signal Process. Syst.* 93, 5 (May 2021), 545–563. doi:10.1007/s11265-021-01640-8
[7] Phillip Karle, Tobias Betz, Marcin Bosk, Felix Fent, Nils Gehrke, Maximilian Geisslinger, Luis Gressenbuch, Philipp Hafemann, Sebastian Huber, Maximilian Hübner, et al. 2023. EDGAR: An Autonomous Driving Research Platform–From Feature Development to Real-World Application. *arXiv preprint arXiv:2309.15492* (2023).
[8] Shinpei Kato, Eijiro Takeuchi, Yoshio Ishiguro, Yoshiki Ninomiya, Kazuya Takeda, and Tsuyoshi Hamada. 2015. An Open Approach to Autonomous Vehicles. *IEEE Micro* 35, 6 (Nov. 2015), 60–68. doi:10.1109/MM.2015.133
[9] Akira Kojima. 2021. Autonomous Driving System implemented on Robot Car using SoC FPGA. In *2021 International Conference on Field-Programmable Technology (ICFPT)*. 1–4. doi:10.1109/ICFPT52863.2021.9609855
[10] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. 2022. Robot Operating System 2: Design, architecture, and uses in the wild. *Science Robotics* 7, 66 (2022), eabm6074. doi:10.1126/scirobotics.abm6074 arXiv:https://www.science.org/doi/pdf/10.1126/scirobotics.abm6074
[11] Víctor Mayoral-Vilches. 2021. Kria Robotics Stack. https://xilinx.github.io/KRS/sphinx/build/html/index.html
[12] Matthew O'Kelly, Varundev Sukhil, Houssam Abbas, Jack Harkins, Chris Kao, Yash Vardhan Pant, Rahul Mangharam, Dipshil Agarwal, Madhur Behl, Paolo Burgio, and Marko Bertogna. 2019. F1/10: An Open-Source Autonomous Cyber-Physical Platform. *CoRR* abs/1901.08567 (2019). arXiv:1901.08567 http://arxiv.org/abs/1901.08567
[13] Richard S. Wallace, Anthony Stentz, Charles E. Thorpe, Hans P. Moravec, William Whittaker, and Takeo Kanade. 1985. First Results in Robot Road-Following. In *International Joint Conference on Artificial Intelligence*. https://api.semanticscholar.org/CorpusID:17211519
[14] Tianze Wu, Yifan Wang, Weisong Shi, and Joshua Lu. 2020. HydraMini: An FPGA-based Affordable Research and Education Platform for Autonomous Driving. In *2020 International Conference on Connected and Autonomous Driving (MetroCAD)*. 45–52. doi:10.1109/MetroCAD48866.2020.00016