2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

SmartKit: User-Friendly Robot with Multiple Operating Systems

Guanyu Chen, Yiqun Zhou, Guoqing Yang, Hong Li and Pan Lv

Abstract— **Mobile robots have become extensively involved in human activities, taking on arduous tasks and providing significant assistance. Robot capabilities have been continuously enhanced, from simple chassis control to path planning and SLAM. Mixed criticality systems enable mobile robots to handle tasks of varying criticality by integrating multiple operating systems, allowing them to accomplish a wide range of tasks. However, besides improving robot computing performance, we should remember that robots are designed to serve humans. Reliability, usability, and affordability are all critical factors for robot design.**

We introduce SmartKit, a mixed criticality system (MCS) for mobile robots. Leveraging the efficiency in hardware utilization brought by virtualization, SmartKit can execute tasks of different criticality efficiently and securely. This paper will present the software and hardware architecture of SmartKit and provide performance and functionality validation of the robot system.

I. INTRODUCTION

With the development of robot systems, mobile robots are gradually playing a significant role in human production activities, providing new solutions to various problems and replacing humans in arduous and complex tasks [1]. Recent progress in AI has sparked widespread discussions and debates about technologies such as embodied intelligence [2], making it even more crucial to focus on mobile robots as an essential carrier of such advancements. Currently, robots are integrating more advanced functionalities into their systems. In addition to traditional control tasks, SLAM and machine learning tasks are being migrated to robot systems [3], [4]. The requirement of powerful system software presents new challenges for robot systems. Some applications must supports realtime features, while others need high performance. Currently, there is no operating system that supports such many features.

Mixed criticality system (MCS), a system composed of various hardware and software components that perform tasks of different criticality, is the key to addressing this challenge [5]. With intergrating different systems, it can compute with high performance while meeting safety certification requirements efficiently. By building an MCS, robots can meet tasks' real-time and performance requirements at the same time. A traditional approach to building an MCS is integrating microcontrollers that run different operating systems using network cables or Controller Area Network (CAN) [6]. However, the requirements of robots, such as size, power consumption, and price,call for MCS to be built more gracefully.

Compared to the traditional method of building the MCS, virtualization is a better solution [7]. In recent years, the development of hardware has made virtualization possible in embedded systems. By virtualizing multiple sets of hardware resources, components of systems with different criticality levels can be easily integrated into the same hardware platform. FlyOS, Bao, and Jailhouse are representative examples of this approach [8], [9], [10]. The research on MCS has entered a new field with the help of hypervisors.

Currently, the research direction of embedded virtualization mainly focuses on improving the performance and security of the hypervisor [11], [12]. Besides these qualities, there are other vital points to consider when designing robot systems. Compared to marginal improvements in computational performance, aspects such as price, usability, and availability are what users can easily experience and differentiate [13]. Therefore, the design of robot software and hardware systems should consider the characteristics above.

Based on the discussions above, we propose SmartKit. SmartKit is a mobile robot based on MCS. Biological tissues inspire the hardware architecture of Smartkit to ensure low power consumption and high performance. Smartkit supports multiple operating systems running on the same hardware. These OSes are managed by SmartVisor, a virtualization platform based on the seL4 [14]. What is more, the SmartVisor is designed to address the requirements of real-world operating scenarios and user needs.

Our contributions can be summarized as follows

- Proposing a mobile robot architecture that is based on bionic structures.
- Introducing a hypervisor-based MCS for mobile robots.
- Designing the SmartVisor, which is aimed to make the system more user-friendly

Sec. II introduces the background and motivation of this work, including an introduction to hypervisor-based MCS and a robot developed by our laboratory. Sec. III introduces the hardware design and software architecture of Smartkit. Sec. IV introduces the essential operating functions of the smartkit and the recovery module designed based on MCS. Sec. V conducts functional and performance testing of the designed system. Sec. VI summarizes the paper and introduces our future work on Smartkit.

II. BACKGROUND AND MOTIVATION

A. Hypervisor-based MCS

Currently, the mixed criticality system is one way in which the robots are evolving [15]. A mixed criticality system

Pan Lv is the corresponding author.

All authors are with the Smart-Vehicle Research Center, Zhejiang University, Hangzhou, China. {pennygrady, 12321119, ygq78, lihong,lvp}@zju.edu.cn

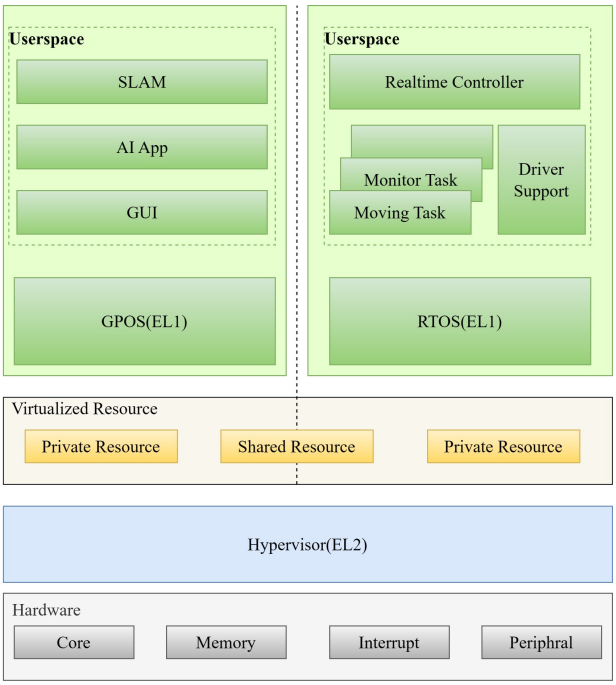

Fig. 1. The general architecture of hypervisor-based MCS

refers to a system that incorporates computer hardware and software capable of executing applications of different criticality levels, such as safety-critical and non-safety-critical, or applications with different Safety Integrity Levels.

Considering the cost associated with development and maintenance, many research groups have proposed utilizing virtualization to construct multi-operating systems (e.g., DriveOS [16], FlyOS [8]). Virtualization can be classified into Type 1 and Type 2 [17]. Type 1 hypervisor, or bare-metal hypervisor, is a software layer installed directly on the physical server and its underlying hardware. In contrast, Type 2 hypervisor, known as hosted hypervisors, runs within the operating system of the physical host machine. Type 1 hypervisors can efficiently manage hardware resources and is more robust than Type 2, which makes them more suitable for embedded systems to design the MCS. There are three main categories for implementing a Type 1 hypervisor: Linux-based, microkernel-based (e.g., QNX [18]), and pure virtualization software (e.g., Bao hypervisor [9]). This paper primarily utilizes the seL4 microkernel for development.

Fig. 1 illustrates a common hypervisor-based MCS architecture on the ARM platform. Through the hypervisor, multiple hardware resources are virtualized into various virtualized resources, which are then allocated to the corresponding guest OSes. The hypervisor ensures proper partitioning of each OS to maintain isolation. However, the hypervisor also provides shared memory to establish communication channels among the guest OSes, facilitating effective collaboration and data exchange within the system. The generous purpose operating system (GPOS) is often used to run complex applications, while the real-time operating system (RTOS) executes the real-time tasks.(e.g., SPHERE [19])

B. Robokit

RoboKit is an innovative mobile robot developed by our laboratory specifically designed for driving tasks. It incorporates a bio-inspired structure and uses embedded artificial intelligence to enhance performance. This unique approach allows the robot to undergo unmanned upgrades without modifying the fundamental structure of conventional devices. As a result, the costs associated with developing and producing mobile robots are significantly reduced. Robo-Sweeper is an unmanned sweeper developed from the Robokit [20].

Robokit comprises two control units: the Automatic-Driving Control Unit (ACU) and the Chassis Control Unit (CCU). The ACU's core computing unit utilizes the Nvidia Jetson Xavier NX module, boasting a computing power of 21 TOPS, capable of meeting the computational requirements for intelligent environment perception, such as radar and image data processing. The CCU adopts the STM32 to fulfill real-time control demands for the mobile robots. The ACU and CCU are interconnected through both Ethernet and CAN, providing redundant control capability.

The main functions of ACU include

- Running laser radar and camera signal processing programs to sense the surrounding environment.
- Running multi-sensor fusion algorithms to fuse IMU, GNSS, and radar data to achieve system positioning.
- Making path planning decisions.

The main functions of CCU include

- Receiving control commands sent from ACU or mobile phone and controlling the intelligent wire-controlled chassis to make corresponding actions.
- Taking charge of collecting chassis data information and providing data information notification and query services for other modules.
- Taking charge of monitoring the vehicle status and providing safety protection in the abnormal or emergency state of the vehicle.

C. Motivation

As mentioned, our laboratory has developed a mobile robot that utilizes multiple hardware components to create a mixed criticality system. Currently, we use Linux on the ACU for perception and decision-making tasks, while an RTOS on the CCU executes the mobile tasks. However, this design approach increases the robot's cost and imposes certain limitations in terms of size and power consumption. Therefore, we have decided to transition our robot into a hypervisor-based mixed criticality system.

After comparing various open-source hypervisors, we have chosen seL4 to implement the virtualization layer. However, we encountered some challenges during the practical application of this hypervisor, such as issues with boot-up speed, virtual machine updating, and functional safety. These problems primarily stem from the current state of hypervisor technology in the embedded domain. Some user requirements, such as high responsiveness, robust security, and user-friendly characteristics, are usually omitted by the

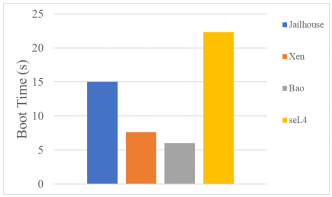

Fig. 2. Boot overhead of several embedded hypervisors

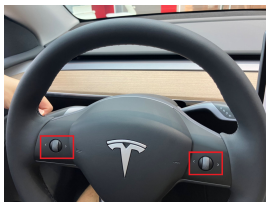

Fig. 3. The manual reset buttons of Tesla Model Y

designer of the embedded systems. However, these features are essential for robots to become famous and usable.

For example, Fig. 2 shows the boot performance of several mainstream embedded hypervisors measured by Martins on ZYNQ [21]. The boot overhead of seL4 is over 20 seconds, much more significant than Bao and Xen. It is unacceptable in mobile robots. The boot delay may cause the system to be unable to handle emergency tasks in time. What is more, in the case of the Tesla Model Y's system crashing, manual intervention is required to restore it, as depicted in Fig.3, with the two buttons used for forcefully rebooting the infotainment system. However, delegating the responsibility of monitoring system operation to the machine can reduce the operational complexity for users and enhance system usability and safety.

Therefore, we have taken steps to improve the system software and transform our robot system into a mixed critical system that is fast, secure, reliable, and easy to update. We focus on addressing the challenges related to boot-up speed, virtual machine updates, and functional safety to provide an enhanced user experience. We aim to promote the broad operation and advancement of robot systems through these improvements.

III. SYSTEM DESIGN

This chapter introduces the hardware and software architecture of SmartKit. The hardware design is still based on Robokit. The most significant difference in the software architecture of SmartKit compared to Robokit is the introduction of a hypervisor called SmartVisor. With the hypervisor, the ACU of SmartKit can simultaneously run multiple operating systems while ensuring isolation between them.

A. Hardware Architecture

Smartkit draws on the structure of the head of higher organisms and proposes a bionic structure for mobile robots. As Fig. 4 shows, This structure integrates complex sensors such as camera, lidar, pickup, GNSS, IMU, and intelligent controller in one Smartkit Head component. The Smartkit Head integrates the camera, lidar, and intelligent controller through Gigabit Ethernet, which makes data transmission convenient. Smartkit Head is installed at the highest point of the equipment and connected to the equipment through the control bus. Through the above architecture design, the following features are realized.

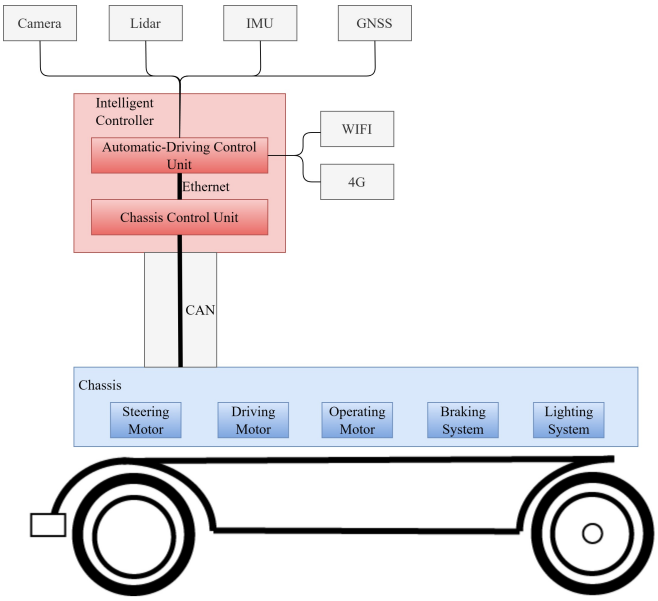

Fig. 4. The hardware architecture of Smartkit

- The camera and lidar sensor are installed in a high position, and the sensing distance is long.
- The upper part of the GNSS sensor is unobstructed, which is convenient for receiving satellite signals.
- The pickup has a 360-degree receiving sound signal Ability.
- The distance between the sensor and the intelligent controller is easy to connect.
- Smartkit Head and the device are only connected through the control bus, signal line, and power line, which is convenient for deployment.

In addition to the Smartkit Head, the Smartkit system also includes the Smartkit Body. The Smartkit Body is built upon a traditional robot and is equipped with a steering motor and an electronic parking mechanism. Its primary function is to execute the steering and parking instructions sent by the Smartkit Head. The motor controller of the robot handles the execution of the forward and brake commands transmitted by the Smartkit Head.

B. System Software

1) SmartVisor: SmartVisor is a virtualization platform developed by our laboratory that is explicitly designed for mobile robots. It utilizes the formally verified seL4 microkernel to develop the virtualization system. By running a cluster of virtual machines on SmartVisor, the platform can provide various computational services with different characteristics to robots. For instance, it can simultaneously run Linux and freeRTOS as virtual machines on SmartVisor, enabling robots to access general-purpose computing and real-time computing services.

The SmartVisor system design comprises two main aspects: downward management of real hardware resources and upward management of virtual machine operations. Downward management involves abstracting and allocating

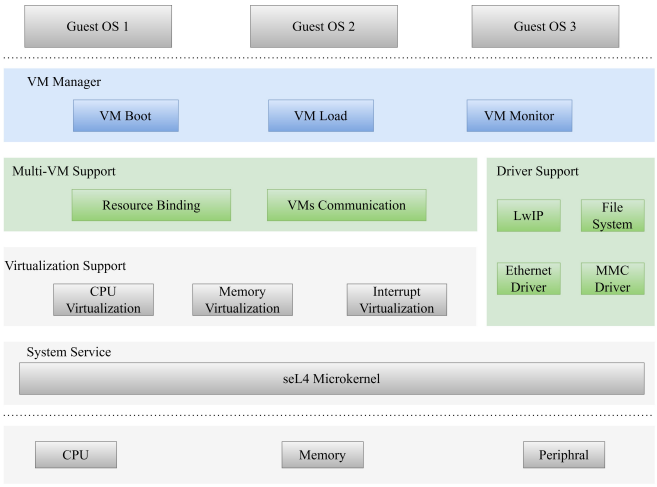

Fig. 5. The architecture of SmartVisor

the underlying hardware resources to ensure that virtual machines can access the required resources. Upward management of virtual machine operations includes monitoring and scheduling the execution of virtual machines to ensure efficient and reliable performance.

Fig. 5 illustrates the system architecture of SmartVisor, which operates between the hardware and operating system layers. It consists of the virtualization support layer, multi-virtual machine runtime support layer, and virtual machine management layer. The gray section in the figure represents open-source code, including the seL4 microkernel, virtualization support, and hardware platform. The blue and green sections represent the work done in this project based on the needs of mobile robots, including the multi-virtual machine runtime support module, driver support module, and virtual machine management module. Our design goal for the hypervisor is to make the mobile robots more user-friendly. To achieve this goal, the SmartVisor needs to be

- **Fast Startup.** Boot overhead is an essential metric for evaluating embedded devices, as shorter boot times allow machines to enter the working state more quickly. In today's fast-paced world, vehicles can ignite and start their applications within a matter of seconds. Therefore, virtualized systems should also meet corresponding standards in their startup process.
- **Flexible Updating.** Updating the Guest OSes is essential. With the fast iteration of the user application, The user and the developer have a solid requirement to update the operating system. However, the virtualization layer replaces the bootloader in these tools, and it is now responsible for updating and loading the OS. In conclusion, Loading guest OS in hypervisor should be as easy as loading OS in bootloader.
- **High Reliability.** Unlike servers, functional safety is important to robot systems. When an operating system crashes, servers can recover their state through restarts or other maintenance operations without posing a significant threat to the safety of the production environment. However, lengthy system recovery times are unacceptable for mobile robots.

2) Multiple OSes: SmartKit is an upgraded version of Robokit. Due to the expenses involved in hardware upgrades, SmartKit has yet to fully migrate all the content from the CCU to the ACU. As a result, the hardware association between the CCU and the chassis unit and the system software on the CCU have been retained. On the ACU, we have deployed two operating systems using SmartVisor. The first is Linux, responsible for executing perception and decision-making tasks. It utilizes the ROS framework, along with various algorithms and drivers, to fulfill the role of the robot's brain. The second operating system on the ACU is SmartOSEK OS, which complies with the AUTOSAR Classic standard. Its role is to detect and recover from any errors in the ACU components. For example, if the Linux guest OS crashes, SmartOSEK OS will promptly detect this situation and initiate recovery. The SmartOSEK OS on the CCU retains its control capabilities over the chassis, allowing it to control the robot's movement based on the commands issued by Linux. Through intelligent middleware, Linux can communicate with both SmartOSEK OS instances on the ACU and CCU.

Overall, SmartKit operates with three operating systems, each fulfilling different responsibilities. The Linux on the ACU handles perception and decision-making tasks, serving as the robot's brain with the help of the ROS framework and various algorithms. The SmartOSEK OS on the ACU is responsible for detecting and recovering from ACU component failures. The SmartOSEK OS on the CCU retains control over the chassis and receives movement commands from Linux. The intelligent middleware facilitates communication between Linux and the SmartOSEK OS instances on the ACU and CCU.

IV. APPLICATION DESIGN

The fundamental functionalities of SmartKit encompass mobility, mapping, and path tracking. This chapter primarily delves into the underlying principles behind implementing these features. Additionally, to ensure the reliability of SmartKit, we have incorporated an additional SmartOSEK OS module on the ACU to serve as a monitoring mechanism.

A. Navigation Algorithm

The process of navigation is as Fig 6 shows. The first run of Smartkit maps everything in the environment using the onboard lidar, with an operator manually controlling the vehicle as it moves through the environment. It creates a comprehensive point cloud map loaded for localization tasks. Waypoints define the trajectories that Smartkit follows on the map. These waypoints are recorded by the robot's localization module when the operator demonstrates the desired path by manually navigating the robot. Smartkit can perform autonomous navigation after the map and the corresponding trajectory are ready.

Iterative closest point (ICP) [22], and Normal Distributions Transform (NDT) [23] are the typical approaches for finding

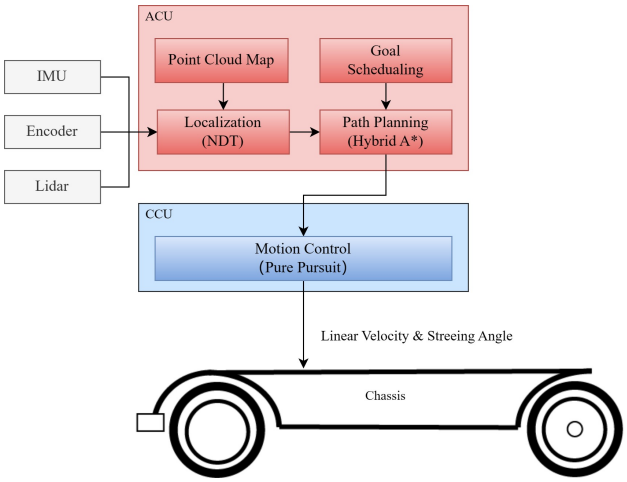

Fig. 6. The execution flow of navigation, from sensor to chassis control

the transformation between two lidar scans. Therefore, we implement a simple system based on these two algorithms for Smartkit's localization stack. Note that ICP has high registration accuracy but suffers from a long runtime when large quantities of points are included in scans. On the other hand, NDT's accuracy is slightly lower. However, it is usually faster than ICP as it reduces the problem's dimensionality using local distribution information instead of point-to-point correspondences. Thus, we employ NDT matching with Newton's iterative method in the front end to obtain the optimized pose transform from the scan to the map. As the robot moves further while building a map, it tends to make small mistakes that add up over time. Loop closure detection and graph-based optimization usually handle map drift problems. For Smartkit, we adopt an algorithm similar to [24] by selecting the nearest frame of the latest keyframe from historical keyframes and using the ICP to see how the two frames match up. If the matching converges and the score is lower than a threshold, we consider the two keyframes similar. Then, the relative pose is used as a constraint to optimize the graph. It maintains the balance between accuracy and computational efficiency in standard conditions.

To perform autonomous navigation, Smartkit is given a list of waypoints as the base trajectory through the environment. The waypoint scheduler identifies the closest remaining waypoint using the vehicle's current pose relative to the map frame. This waypoint then serves as the local goal for Smartkit's path-planning algorithms. Operations in practical scenarios require UGVs to do more than follow the base trajectory. Therefore, we use the 2D grid-based cost map mechanism implemented in ROS to represent the surrounding environment for obstacle avoidance. The latest point cloud captured by the lidar is used as the observation source of the costmap after removing its ground point cloud with a ray ground filter [25] and discarding far-away points. To generate feasible trajectories, the geometric details of paths, as well as the vehicle's kinematic constraints, should be taken into account [23]. Smartkit's navigation stack receives the goal and uses a hybrid A* [26] planner to compute a desired local path with a simplified vehicle configuration. The vehicle follows this path using the pure pursuit [27]algorithm, which calculates the linear velocity and the steering angle.

B. Recovery Module

The system recovery module is designed to address the Linux system crashes during robot operation. The Linux crash can pose a significant risk to the system. SmartKit utilizes SmartOSEK OS to monitor Linux status.

We designed a typical monitoring module between two guest OSes. As SmartOSEK OS has a shorter startup time than Linux, it usually initiates the activation of the monitoring module. It sends a data packet containing a characteristic value to Linux every 2 seconds. Once Linux completes its startup and receives the data packet, it responds with an acknowledgment packet to SmartOSEK OS. Through this handshake process, the monitoring module becomes fully operational.

Linux sends a heartbeat packet to SmartOSEK OS every 1 second to confirm its liveliness. The heartbeat packet includes a characteristic value processed by an algorithm to verify its correctness and ensure proper execution. Currently, the algorithm implementation increments the characteristic value with each transmission. SmartOSEK OS checks for the heartbeat packet's presence and verifies the characteristic value's correctness every 1 second. When errors accumulate to a specified threshold, SmartOSEK OS triggers the recovery operation for the Linux client and restarts the monitoring module by requesting a synchronized data packet.

Through this process, the SmartKit system recovery module actively monitors the Linux system, ensuring its proper operation and taking necessary action in case of failures or crashes. Currently, the system will simply reboot the crash component when Linux crashes.

V. EVALUATION

We conducted practical scenario tests on SmartKit to verify the interaction between RoboKit and SmartVisor, as well as the advanced features enabled by SmartKit, specifically the crash detection and recovery functions during robot running. For details on the hardware performance of SmartKit and the software performance of SmartVisor, please refer to our previous work [28]. Here we mainly introduce our work of the moving function.

A. System Configuration

Table I and Table. II respectively introduce the hardware environment and software configuration of Smartkit. These are the basic configurations required for smartkit to complete its operational tasks. To enable the Smartkit to have the ability to perceive the environment and drive autonomously, we installed Smartkit Head for the robot system. The installation position is above the steering wheel. It has a camera, 16-line lidar, GNSS, IMU, encoder, etc. The SmartViosr will run on the Nvidia Jetson Xavier NX. It includes a hexa-core

TABLE I

HARDWARE PARAMETER

Processor related	
ACU	Nvidia Jetson Xavier NX
CCU	STM32F407
GPU	384-core NVIDIA Volta GPU
RAM	8GB
Storage	16GB
Sensor related	
Lidar	16-wire mechanical type
Camera	1080p Camera x3
Encoder	Absolute encoder
IMU	Nine-axis inertial navigation unit
GNSS	GPS (Meter-level accuracy)/BEIDOU

TABLE II

SOFTWARE CONFIGURATION

Guest Linux	L4T-Linux 32.5.1
CPU number	0-3
Memory size	6GB
Guest RTOS	SmartOSEK OS
CPU number	4
Memory size	256MB

NVIDIA Carmel, a GIC-400 featuring 4 list registers and an MMU-500 (SMMUv2). Cores have private,and share a L2 6MiB unified cache and a L3 4MB unified cache. In the testcase, the Linux and the RTOS will run independly and run at the same time. The version of Linux kernel is L4T Linux 32.5.1. This guest OS is assigned with 4 CPUs and 6GB memory, while the RTOS named SmartOSEK OS is assigned 1 CPU and 256MB memory.

B. Moving in Garage

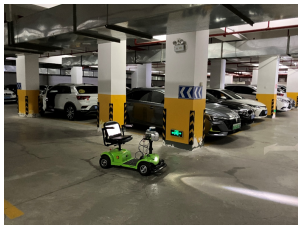

Fig. 7. The test environment for Smartkit

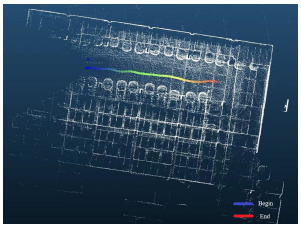

Fig. 8. The pointcloud of the environment collected by Smartkit

To test Smartkit, we put the robot into the underground garage and executed the moving tasks shown in Fig. 7. We remotely logged into SmartKit via a Wi-Fi network and controlled it to move in a straight line for a certain distance within the garage. The SmartKit program recorded vehicle parameters and environmental information during the operation.

Fig. 8 shows a point cloud top view of the underground parking garage and the robot's movement trajectory in the scene. The robot scans the surrounding environment using an integrated laser radar, and mapping and localization are performed based on the point cloud data. The figure

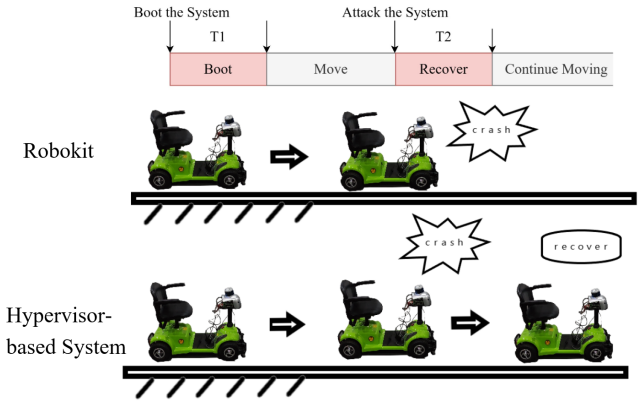

Fig. 9. Different actions from two systems when the Linux is down

demonstrates that point cloud data can accurately describe the environmental information of the underground parking garage.

C. Recovery during Moving

Besides the simple moving task, we conducted a system crash test for robots, including Robokit,seL4-based Robokit, and Smartkit.seL4-based Robokit has the same architecture as the Smartkit. But the hypervisor is the origin seL4 hypervisor which is not optimized by us. As depicted in Fig. 9, we set up a script to automatically start running 20 seconds after the system is powered on. Once the system is initialized, the robot will move after 20 seconds. Using a timed script, we intentionally put the Linux system into a crash state during the robot's movement. As a result, Robokit will stop running due to the Linux crash, while a hypervisor-based system will detect the crash and initiate the recovery process. Once the recovery is completed, the hypervisor-based system will continue with its movement task.

Based on our process analysis, we focused on two main durations: T1, the time from power-on to system initialization, and T2, the time for the hypervisor-based system to recover from the crash and resume operation. The experimental results are shown in Table. III. T1 includes the boot loader startup time and script launch time, where the boot loader startup overhead depends on the specific experimental platform and configuration. The script launch time was fixed at 15 seconds. Comparing these systems, SmartKit incurs additional startup overhead from the hypervisor, but it is much less than the seL4 hypervisor because of our optimization for seL4. Due to the impact of virtualization on virtual machine performance, the startup time for the Linux virtual machine is increased by 2.65 seconds compared to bare metal. Regarding T2, the system recovery time is 25.13 seconds. Through program analysis, we found that the overhead mainly comes from the recycling of seL4 capabilities and the restart of the Linux virtual machine. The monitoring overhead is less than 1 second and can be omitted. Therefore, the overhead of seL4 capability recovery

is approximately 6 seconds, which needs to be optimized in future work.

TABLE III
BOOT-RECOVER OVERHEAD

Boot overhead(s)		Robokit	seL4-based Robokit	Smartkit
T1	Bootloader	26.36	26.36	26.36
	Hypervisor	-	48.91	9.39
	Linux	16.89	19.54	19.54
T2	Reovery	-	25.13	25.13

VI. CONCLUSION AND FUTURE WORK

This article introduces SmartKit, a hybrid critical system based on virtualization technology. Its primary goal is to make mobile robot systems more user-friendly. Through intelligent virtualization technology and the simultaneous operation of multiple operating systems, SmartKit's startup performance and reliability have been greatly improved.

Although significant progress has been made, some tasks must be completed. Firstly, we plan to gradually migrate the system software from the CCU to the ACU, gradually phasing out the CCU to reduce power consumption and costs in the robot system. Secondly, with the increasing popularity of FPGA (Field-Programmable Gate Array) technology, we plan to port the system software to an FPGA-based SoC in the future, replacing the ACU. These improvements will further enhance the performance and functionality of SmartKit, making it a more advanced and reliable mobile robot system.

ACKNOWLEDGMENT

This work was supported by the Natural Science Foundation of China (No.U22A202101) and the Pioneer and Leading Goose R&D Program of Zhejiang (No.2024C01016)

REFERENCES

[1] Q.-n. Zhang, F.-f. Zhang, and Q. Mai, "Robot adoption and labor demand: A new interpretation from external competition," *Technology in Society*, vol. 74, p. 102310, 2023.

[2] N. Roy, I. Posner, T. Barfoot, P. Beaudoin, Y. Bengio, J. Bohg, O. Brock, I. Depatie, D. Fox, D. Koditschek *et al.*, "From machine learning to robotics: Challenges and opportunities for embodied intelligence," *arXiv preprint arXiv:2110.15245*, 2021.

[3] P. Jiang, D. Ergu, F. Liu, Y. Cai, and B. Ma, "A review of yolo algorithm developments," *Procedia Computer Science*, vol. 199, pp. 1066–1073, 2022.

[4] T. Taketomi, H. Uchiyama, and S. Ikeda, "Visual slam algorithms: A survey from 2010 to 2016," *IPSJ Transactions on Computer Vision and Applications*, vol. 9, no. 1, pp. 1–11, 2017.

[5] A. Burns and R. Davis, "Mixed criticality systems-a review," *Department of Computer Science, University of York, Tech. Rep*, pp. 1–69, 2013.

[6] W. Steiner and G. Bauer, "Mixed-criticality networks for adaptive systems," in *29th Digital Avionics Systems Conference*. IEEE, 2010, pp. 5–A.

[7] G. J. Popek and R. P. Goldberg, "Formal requirements for virtualizable third generation architectures," *Communications of the ACM*, vol. 17, no. 7, pp. 412–421, 1974.

[8] A. Farrukh and R. West, "Flyos: Integrated modular avionics for autonomous multicopters," in *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2022, pp. 68–81.

[9] J. Martins, A. Tavares, M. Solieri, M. Bertogna, and S. Pinto, "Bao: A lightweight static partitioning hypervisor for modern multi-core embedded systems," in *Workshop on next generation real-time embedded systems (NG-RES 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[10] R. Ramsauer, J. Kiszka, D. Lohmann, and W. Mauerer, "Look mum, no vm exits!(almost)," *arXiv preprint arXiv:1705.06932*, 2017.

[11] R. Russell, "virtio: towards a de-facto standard for virtual i/o devices," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 95–103, 2008.

[12] A. Golchin, S. Sinha, and R. West, "Boomerang: Real-time i/o meets legacy systems," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2020, pp. 390–402.

[13] T. Lourens and E. Barakova, "User-friendly robot environment for creation of social scenarios," in *International Work-Conference on the Interplay between Natural and Artificial Computation*. Springer, 2011, pp. 212–221.

[14] K. Elphinstone and G. Heiser, "From l3 to sel4 what have we learnt in 20 years of l4 microkernels?" in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, 2013, pp. 133–150.

[15] F. Lumpp, F. Fummi, H. D. Patel, and N. Bombieri, "Enabling kubernetes orchestration of mixed-criticality software for autonomous mobile robots," *IEEE Transactions on Robotics*, 2023.

[16] S. Sinha and R. West, "Towards an integrated vehicle management system in driveos," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 20, no. 5s, pp. 1–24, 2021.

[17] J. Vojtesek and M. Pipis, "Virtualization of operating system using type-2 hypervisor," in *Software Engineering Perspectives and Application in Intelligent Systems: Proceedings of the 5th Computer Science On-line Conference 2016 (CSOC2016), Vol 2 5*. Springer, 2016, pp. 239–247.

[18] D. Hildebrand, "An architectural overview of qnx." in *USENIX Workshop on Microkernels and Other Kernel Architectures*. Citeseer, 1992, pp. 113–126.

[19] A. Biondi, D. Casini, G. Cicero, N. Borgioli, G. Buttazzo, G. Patti, L. Leonardi, L. L. Bello, M. Solieri, P. Burgio *et al.*, "Sphere: A multi-soc architecture for next-generation cyber-physical systems based on heterogeneous platforms," *IEEE Access*, vol. 9, pp. 75 446–75 459, 2021.

[20] G. Chen, P. Lv, H. Li, and G. Yang, "Robo-sweeper: Bionics based unmanned sweeper platform," in *2021 IEEE 23rd Int Conf on High Performance Computing & Communications; 7th Int Conf on Data Science & Systems; 19th Int Conf on Smart City; 7th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*. IEEE, 2021, pp. 1381–1388.

[21] J. Martins and S. Pinto, "Shedding light on static partitioning hypervisors for arm-based mixed-criticality systems," in *2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2023, pp. 40–53.

[22] P. J. Besl and N. D. McKay, "Method for registration of 3-d shapes," in *Sensor fusion IV: control paradigms and data structures*, vol. 1611. Spie, 1992, pp. 586–606.

[23] M. Magnusson, A. Lilienthal, and T. Duckett, "Scan registration for autonomous mining vehicles using 3d-ndt," *Journal of Field Robotics*, vol. 24, no. 10, pp. 803–827, 2007.

[24] Y. Jiang, T. Wang, S. Shao, and L. Wang, "3d slam based on ndt matching and ground constraints for ground robots in complex environments," *Industrial Robot: the international journal of robotics research and application*, vol. 50, no. 1, pp. 174–185, 2023.

[25] M. Himmelsbach, F. V. Hundelshausen, and H.-J. Wuensche, "Fast segmentation of 3d point clouds for ground vehicles," in *2010 IEEE Intelligent Vehicles Symposium*. IEEE, 2010, pp. 560–565.

[26] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Path planning for autonomous vehicles in unknown semi-structured environments," *The international journal of robotics research*, vol. 29, no. 5, pp. 485–501, 2010.

[27] R. C. Coulter *et al.*, *Implementation of the pure pursuit path tracking algorithm*. Carnegie Mellon University, The Robotics Institute, 1992.

[28] G. Chen, P. Lv, H. Li, and G. Yang, "Smartvisor: User-friendly hypervisor for mobile robots," in *Proceedings of the 25th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*, 2024, pp. 62–71.