Hindawi
Journal of Robotics
Volume 2021, Article ID 6656912, 10 pages
https://doi.org/10.1155/2021/6656912

Research Article

A Delivery Robot Cloud Platform Based on Microservice

Zuozhong Yin,[1,2] Jihong Liu,[1] Bin Chen,[2] and Chuanjun Chen[3]

[1]*School of Mechanical Engineering & Automation, Beihang University, Beijing 100191, China*
[2]*Beijing Research Institute of Automation for Machinery Industry Co., Ltd., Beijing 100120, China*
[3]*RIAMB (Beijing) Technology Development Co., Ltd., Beijing 100120, China*

Correspondence should be addressed to Bin Chen; chenbin_vips@163.com

Received 4 December 2020; Revised 21 January 2021; Accepted 4 February 2021; Published 18 February 2021

Academic Editor: Giovanni Muscato

Delivery robots face the problem of storage and computational stress when performing immediate tasks, exceeding the limits of on-board computing power. Based on cloud computing, robots can offload intensive tasks to the cloud and acquire massive data resources. With its distributed cluster architecture, the platform can help offload computing and improve the computing power of the control center, which can be considered the external "brain" of the robot. Although it expands the capabilities of the robot, cloud service deployment remains complex because most current cloud robot applications are based on monolithic architectures. Some scholars have proposed developing robot applications through the microservice development paradigm, but there is currently no unified microservice-based robot cloud platform. This paper proposes a delivery robot cloud platform based on microservice, providing dedicated services for autonomous driving of delivery robot. The microservice architecture is adopted to split the monomer robot application into multiple services and then implement automatic orchestration and deployment of services on the cloud platform based on components such as Kubernetes, Docker, and Jenkins. This enables containerized CI/CD (continuous integration, continuous deployment, and continuous delivery) for the cloud platform service, and the whole process can be visualized, repeatable, and traceable. The platform is prebuilt with development tools, and robot application developers can use these tools to develop in the cloud, without the need for any customization in the background, to achieve the rapid deployment and launch of robot cloud service. Through the cloud migration of traditional robot applications and the development of new APPs, the platform service capabilities are continuously improved. This paper verifies the feasibility of the platform architecture through the delivery scene experiment.

1. Introduction

In 2020, the COVID-19 pandemic unraveled the weak points in the global supply chain for goods. With the rapid development of autonomous driving technology, the application scenarios of wheeled robots are more and more extensive. Companies such as Amazon and logistics and supply chain management organizations have been experimenting with wheeled robots to deliver their own packages and have developed unmanned delivery robots. As application scenarios become more complex, the computing power requirements of delivery robots are also increasing.

Cloud robot technology is an emerging technology developed by robots with the help of cloud computing. The storage and computing power of traditional robots are limited to the robot body, while intelligence requires more knowledge storage, retrieval, and reasoning computing capabilities. In response to this question, Professor James Kuffner of Carnegie Mellon University in 2010 first proposed the concept of cloud robots [1–3]. A cloud robot is a combination of robotics technology and cloud computing; it offloads complex computing functions, such as the data processing, planning, decision-making, and collaboration of robots, to the cloud. As long as the robot is equipped with simple sensors and connected to the network, it can complete complex tasks. The robot cloud platform is a software service platform that combines robot technology and Internet technology to provide users with professional services such as network-based robot access, monitoring, management, data analysis, and control optimization; it can provide

the computing power of the entire cloud to each robot, strengthen the memory and judgment level of the robot, deploy the development and test environment, program the robot as a service in the cloud, and innovate the robot development process. Deploying services in the cloud allows robots to transform functionality by changing their configuration in the cloud to improve robot usability.

In 2010, Singapore's ASORO Lab studied DAvinCi [4], a framework for running the FastSlam algorithm in the Hadoop platform. The DAvinCi software framework is based on the Hadoop cluster, combined with the robot operating system (ROS) as the message delivery framework of the robot ecosystem [5], using the platform as the "brain" of the robot, using cloud computing to provide scalability and parallelism for the service robot, which can be regarded as a prototype of the robot cloud platform. In 2011–2014, the RoboEarth [6] project was launched in Europe. RoboEarth has evolved into a cloud-based database in which robots can share information about the environment, tasks, and objects [7]. In 2012, Doriya [8, 9] proposed a cloud computing framework, the Robot cloud, to provide computing services for robots. In 2015, Mohanarajah et al. [10] developed a Rapyuta cloud robot platform based on RoboEarth, enabling robots to share knowledge and learn from each other through access to the RoboEarth knowledge base. Rsi-Cloud [11] is a networked robotic system that combines the RoboEarth project's approach with UNR-PF's distributed execution capabilities and provides a platform for robot service providers and robot developers to develop robot services and applications separately. In 2017, Zhou et al. proposed an overall architecture of a robot cloud platform based on SOA. In 2018, Amazon provided the public with a ROS cloud robot development platform AWS RoboMaker to help users easily develop, test, and deploy robot applications. Xia proposed a new lightweight cloud robot architecture based on microservice. In 2019, a novel robot cloud platform called cloud robotics intelligent cloud platform (CRICP) [12] is designed by Liu to overcome the problem that heterogeneous service robots cannot access the cloud platform.

The delivery robot cloud platform based on microservice proposed in this paper divides robot applications into microservices, which reduces the coupling degree of platform software and the difficulty of code maintenance. Due to the automated deployment components, the deployment time of platform service is obviously better than that of single architecture and SOA-based robot cloud platform, which greatly reduces the cost of operation and maintenance.

2. Needs of the Delivery Robot Cloud Platform

2.1. Calculate Unloading Needs. Robot unloading computing depends on the cloud platform to achieve. Cloud platform, also known as hardware virtualization, abstract and transform the physical hardware resources of many computers, which can be divided and combined into one or more computer configuration environments. Thus, breaking the impartible barrier between entity constructs enables users to use these computer hardware resources in a better form than the original configuration. Virtualization is a way to complete the pooling of hardware resources. Mainstream cloud virtualization technologies and cloud platform management systems include EXSI technology in VMware software, Microsoft Hyper-V products, open-source Xen, and OpenStack, etc.

2.2. Service Deployment Needs. The modern robot control system is usually a logical design for a distributed system based on the component; each unit can abstract some hardware parts or functions, and opening to the rest of the system running a standard interface to build and deliver complex ROS applications and services for nonprofessionals can be a difficult task. Which software architecture should be used to rapidly deploy ROS packages is a key consideration for the platform.

2.3. Architecture Design. The microservice architecture is a cloud-native architecture whose goal is to split the application into a series of small microservices, each of which can be deployed independently on potentially different platforms and technology stacks. Originally, the web was primarily considered a means of presenting information to a wide range of people, but SOA programming led to a fundamental shift from web to computing architecture, and SOA brought a paradigm shift in methodology when designing and implementing distributed systems. In this architecture, the system is broken down into integrated services. The microservice architecture is a specialized approach to SOA implementation [13], emphasizing lightweight virtual machines, where applications are developed as a collection of fine-grained services that run as separate processes and decompose the entire functionality of the application into a set of services that can be deployed and extended independently, with each service completing one job.

The DAvinCi cloud computing platform is an important attempt to combine the advantages of robotics with cloud computing. It proves that the execution time of executing algorithms in the cloud to build large-area maps can be significantly shortened, which greatly improves the performance of robots. However, its architectural design is mainly aimed at this specific scenario of real-time positioning and map construction, and its versatility is not strong. The RoboEarth architecture does not take into account the problem of how to deploy services in the cloud, and how to improve the efficiency of resource usage is the difficulty of the system. The overall architecture of the service robot cloud platform uses a service-oriented architecture (SOA) to build robot cloud services, focusing on the scheduling and management of cloud platform services. But the SOA integration mechanism and centralized governance predetermined the bottleneck when the system needs to be expanded, so the microservice architecture seems to be ready to replace SOA as the dominant industrial architecture. The microservice-based cloud robotics system for intelligent space [14] proposes adopting microservice architecture to develop robot applications but fails to solve the problem of

automatic construction of cloud platform robot services. Therefore, this paper will focus on describing the architecture and implementation of the delivery robot cloud platform based on microservice, as well as the process of automatic orchestration and deployment of cloud services.

3. Delivery Robot Cloud Platform Based on Microservice

As shown in Figure 1, the delivery robot cloud platform based on microservice is divided into four layers: from bottom to top is the physical layer, communication interface layer, micro application layer, and business layer.

3.1. Physical Layer. The physical layer is the hardware hierarchy that provides infrastructure resources, including servers, storage, and network resources. It can be based on the public cloud IaaS layer or build its own private cloud to provide users with IaaS services and deploy tools such as Kubernetes (k8s) and Docker, which solves the problems of virtualization and automatic management of IT resources.

The physical layer in Figure 1 of the cloud platform adopts a private cloud. The physical server, storage device, and network resources are integrated and virtualized by OpenStack [15], thereby completing the dynamic allocation of resources and realizing the horizontal expansion of the platform to achieve hardware customization and flexible management. The deployment of OpenStack must be tailored to the demands to support users to host compute-intensive tasks to the cloud platform. The autonomous driving cloud platform draws on the loosely coupled architecture of OpenStack to complete the design of the physical layer and selects the corresponding components according to the requirements of the cloud platform to provide support for the upper application services.

After the platform is virtualized, the Docker container is installed to isolate the resources, and the k8s is installed to orchestrate and deploy the containers, which can realize automated operation and maintenance management of the robot cloud platform.

3.2. Communication Interface Layer. The communication layer preinstalls the Ubuntu system and development tools and integrates the ROS to realize the interaction between the robot and the cloud platform. ROS is not a traditional operating system, such as Windows, Linux, and MacOS, but provides a cross-platform modular software communication mechanism and software development framework. The topic-based communication mechanism implemented by ROS provides communication support functions for robots, which decouples the logic between different applications (nodes), so it is widely applied to the actual development of robot software services. When the ROS system version of the cloud platform is consistent with the robot, remote communication can be achieved through ROS_MASTER.

3.3. Micro Application Layer. The micro application layer consists of a number of functionally independent and well-defined service particles that communicate with a lightweight restful protocol. They are dynamically reorganized in real time and are an important resource that constitutes user layer services. The following components are required to implement governance of microservices.

3.3.1. Zuul. Zuul is a microservice API gateway that provides dynamic routing, monitoring, resilient load, and security capabilities for accessing and invoking microservices. It is a front-facing portal entry in the overall network system.

3.3.2. Consul. Consul is used to implement the registration and discovery of microservices. Service providers typically provide services in clusters and notify service callers of the service address so that they can discover the target service.

3.3.3. Ribbon. The ribbon is used to achieve client load balancing, by positioning the middle-tier services running in the AWS domain, to achieve the purpose of load balancing and middle-tier service failover.

3.3.4. Message Bus. Message bus is used to implement communication between microservices. Message bus integrates event processing mechanism and message middleware to send and receive messages, which are mainly composed of the sender and the receiver and the event. For different business requirements, different events can be set, the sender sends the event, and the receiver accepts the corresponding event and handles the corresponding event.

3.3.5. Config Center. The distributed config center is used to manage profile uniformly, making deployment and maintenance easier. The profile can be centralized in the GitHub repository, and a new configuration server can be created to manage all profiles. When a configuration change is required during maintenance, it is simply pushed to a remote repository after the local change.

3.4. Business Layer. The business layer encapsulates resource abstraction and virtualization into services deployed on the cloud platform to provide software services to users, which is the top level of the cloud. The cloud platform provides common services such as offline calculation, data storage, and map drawing for wheeled robots [16]. It also provides dedicated services for autonomous driving, such as autonomous parking, automatic following, and lane keeping. Through the offline calculation of the cloud platform, the computing demands in the autonomous driving process are solved. The data service processes the sensor data uploaded by the robot; the cloud receives the information read by the robot through the sensor and then performs heterogeneous data fusion, machine learning, analysis, and sharing to the robot that is about to reach the area.

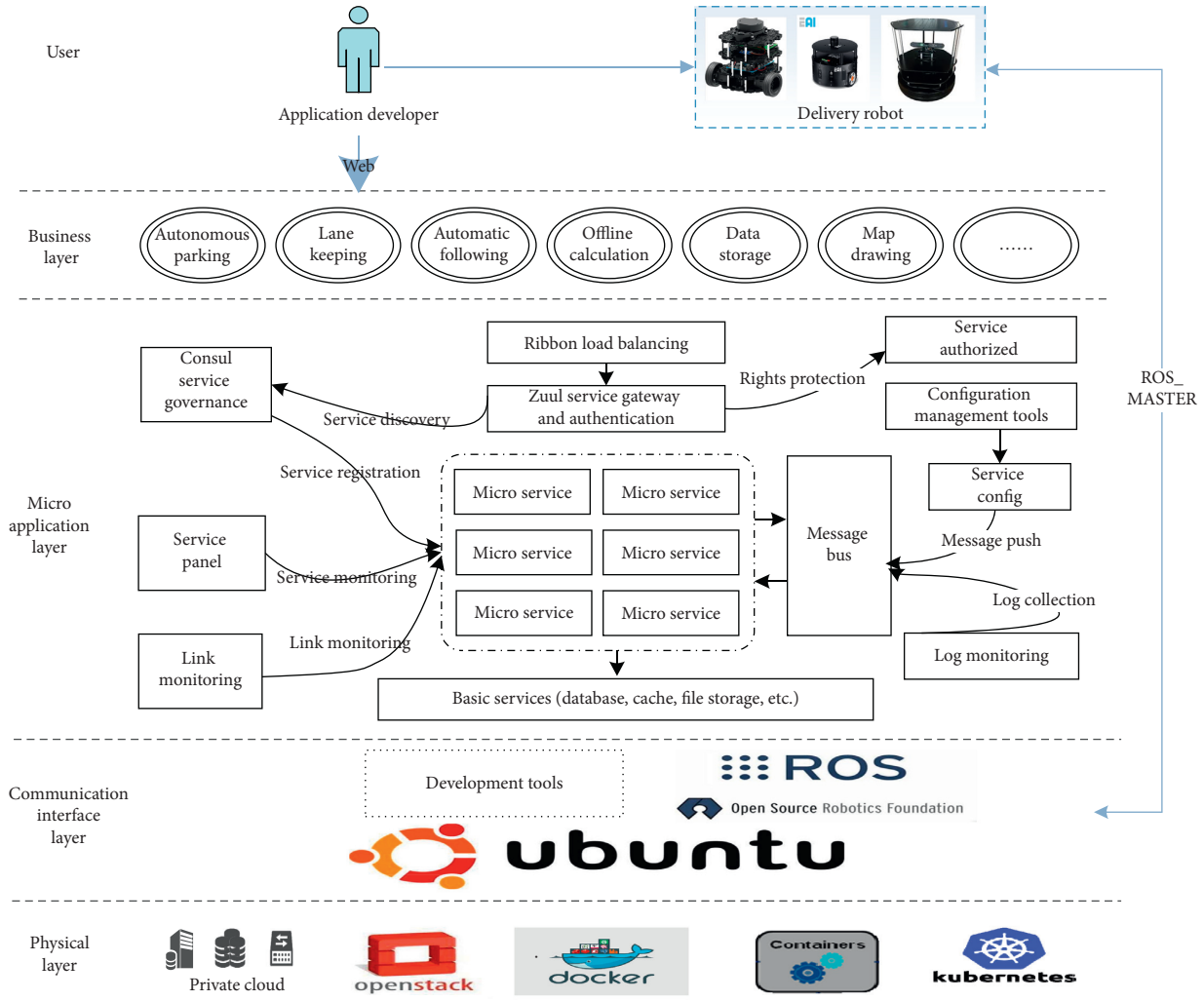


Figure 1: Delivery robot cloud platform based on microservice.

4. CI/CD of Robot Cloud Platform

4.1. CI/CD Component. The cloud platform adopts the architecture of microservice to design the encapsulation of services, and Docker is used to build the microservices of the platform. The CI/CD (continuous integration, continuous deployment, and continuous delivery) of cloud platform services is implemented by a series of components such as k8s, Jenkins, Harbor, and Pipeline [17].

4.1.1. Kubernetes. Kubernetes (K8s) is a service-oriented portable container orchestration management tool for automatic deployment, extension, and management of containerized applications, making the deployment, operation, and maintenance of our applications more convenient.

4.1.2. Jenkins. Jenkins is an open-source, extensible, continuous integration, delivery, deployment (software/code compilation, packaging, build, test, and deployment) based web interface platform. It can upload the code to the warehouse (such as GitHub, GitEE, and GitLab) and then automatically deploy the latest code from the code warehouse in the web page, rather than manually packaging, uploading the server, and deploying the series of steps, which is very convenient.

4.1.3. Harbor. Harbor is an enterprise-class private registry server with capabilities such as authority management (RBAC), LDAP, administrative interface, self-registration, and mirror replication. Harbor is Docker's mirrored repository, providing a layered transport mechanism to optimize network transport, provide a web interface to optimize user experience, and support horizontal cluster expansion, with a good security mechanism.

4.1.4. Pipeline. Pipeline is an official Jenkins plug-in that can be used to implement and integrate continuous delivery within Jenkins. Pipeline is a process that defines the steps to complete a CI/CD process. Instead of completing a CI/CD process manually and automatically, the process is user-defined.

4.2. CI/CD Process. As shown in Figure 2, the code of the algorithm is sent to the GitLab warehouse. Jenkins pulls the code, executes the code compilation, builds the mirror, and then pushes the code to the Harbor repository. Then, Docker pulls the code from the repository, delivers it to the production environment, and uses k8S to orchestrate the containers to achieve high availability and clustered distribution of resource services. The robot microservice particles are distributed in a container by k8s. Jenkins calls the k8s API to dynamically create Pods. Pod is the smallest unit in k8s and is a combination of a group of containers. Once created, Pods automatically extract code, build images and push them, and then automatically destroy and release resources when they complete the task. Robot application developers can pull images from the Harbor repository, deploy them to any environment, and give users access.

5. Workflow of Platform in Delivery Scene

Experimental setup: deploy microservices in the cloud and deliver prop to the designated parking spot. The robot calls the cloud service and completes a series of operations in real time. The algorithm is split and deployed on the platform as shown hereinafter.

The operation flow of autonomous parking is shown in Figure 3.

(1) The parking environment is detected by sensors such as cameras, and the parking sign and parking space are detected

(2) According to the information uploaded by the sensor, the effective parking space information and the relative position of the vehicle are obtained, thereby determining the initial position of the parking

(3) The control unit models the real-time environment based on the sensor information, generates a motion path, and controls the wheeled robot to automatically move to the parking space without collision

The key technologies involved in autonomous parking include lane detection, sign identification, and motion control. This paper uses an open-source module "Autorace." These algorithms are converted into microservices and split into the detect_lane module, detect_parking module, detect_sign module, control_lane module, control_parking module, and control_decider module. Each module is a subsystem and a microservice that constitutes an autonomous parking cloud service. Each microservice is a separate Java process running on a separate virtual machine (container). A single microservice can be developed independently, can use different development languages, and is easier to deploy into the proposed cloud platform than the traditional monolithic architecture.

The delivery robot used in the experiment is shown in Figure 4. Turtlebot3 is a new generation of ROS open-source wheeled robot platform newly launched by Korea's ROBOTIS.

The information flow between the robot and the cloud platform is shown in Figure 5 and can be described by the following steps:

Task 1: Log in to the background interface of the cloud platform, remotely start up the robot by terminal command, establish the connection between the robot and the cloud platform, and collect the running data of the robot in real time. The ROS system uploads the environmental information collected by the robot lidar and camera to the cloud. As shown in Figure 6, the map of the whole laboratory is first established through the lidar of the robot, and then the target position is sent. In the experiment, the IP address of the cloud host was 10.1.12.29, which could be accessed through a web browser.

Task 2: Through Zuul Gateway identity authentication, access to the microservices deployed on the platform, and support traffic scheduling.

Task 3: Locate the microservice information registered in the Consul component, and ensure high availability of services through TCP/IP calls between microservices.

Task 4: Compute in the cloud, and then feedback the calculation result to the robot for execution.

As shown in Figure 7, when the delivery robot enters the lane, it needs to deliver the prop to a designated parking spot. The steps for the robot to call the microservice are as follows: (1) when the robot is traveling on the lane, detect_sign service is called to detect the stop sign; when a stop sign is detected, detect_lane service is called to detect lane lines; (2) when dashed lane lines are detected, control_lane and detect_parking services are called to find the correct parking spot; (3) when a parking spot is detected, the control_decider and control_parking services are called to park the robot in the parking spot.

6. Discussion

The experiment of the delivery scene proves that a series of processes such as automatic deployment and invocation of the platform service can be realized. As shown in Table 1, the platform scheme proposed in this paper has obvious advantages in software development and deployment compared with other schemes. It can be seen from Table 1 that the time required for platform service deployment in the solution proposed in this paper is significantly better than the other two solutions. Due to the need to divide deployed applications into microservices, they have lower coupling degree and easier code maintenance, which can reduce operation and maintenance costs, and the platform has better scalability [14].

In order to verify the service invocation performance of the proposed delivery robot cloud platform based on microservice, we performed service invocation experiments on the monomer CR system based on ROS, SOA-based robot cloud platform, and the proposed delivery robot cloud platform based on microservice. Service invocation

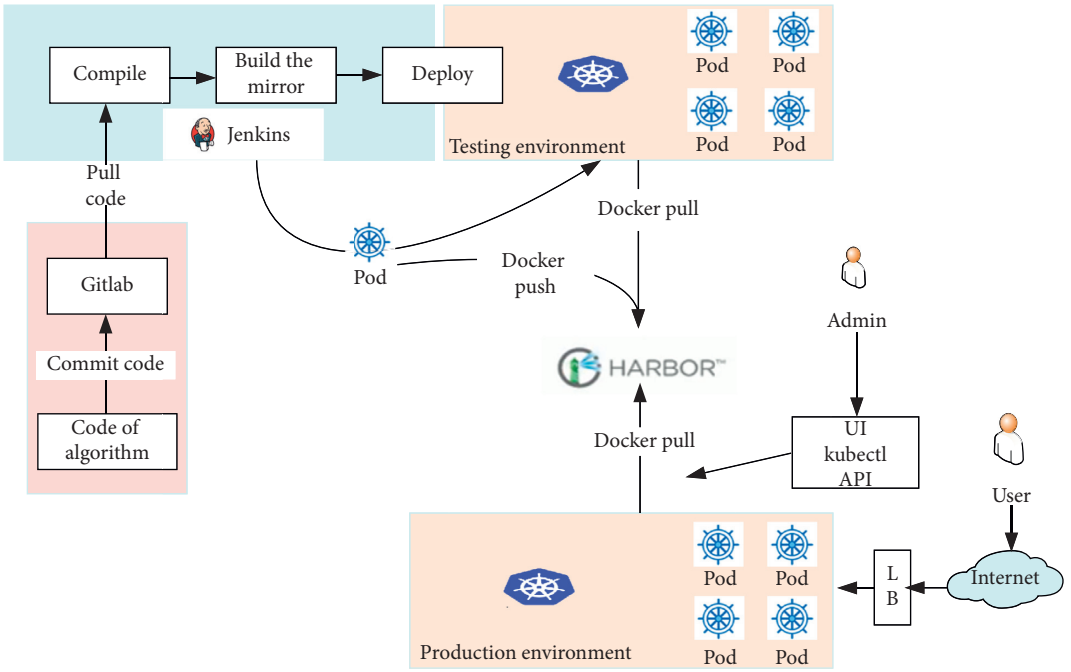


Figure 2: CI/CD of code.

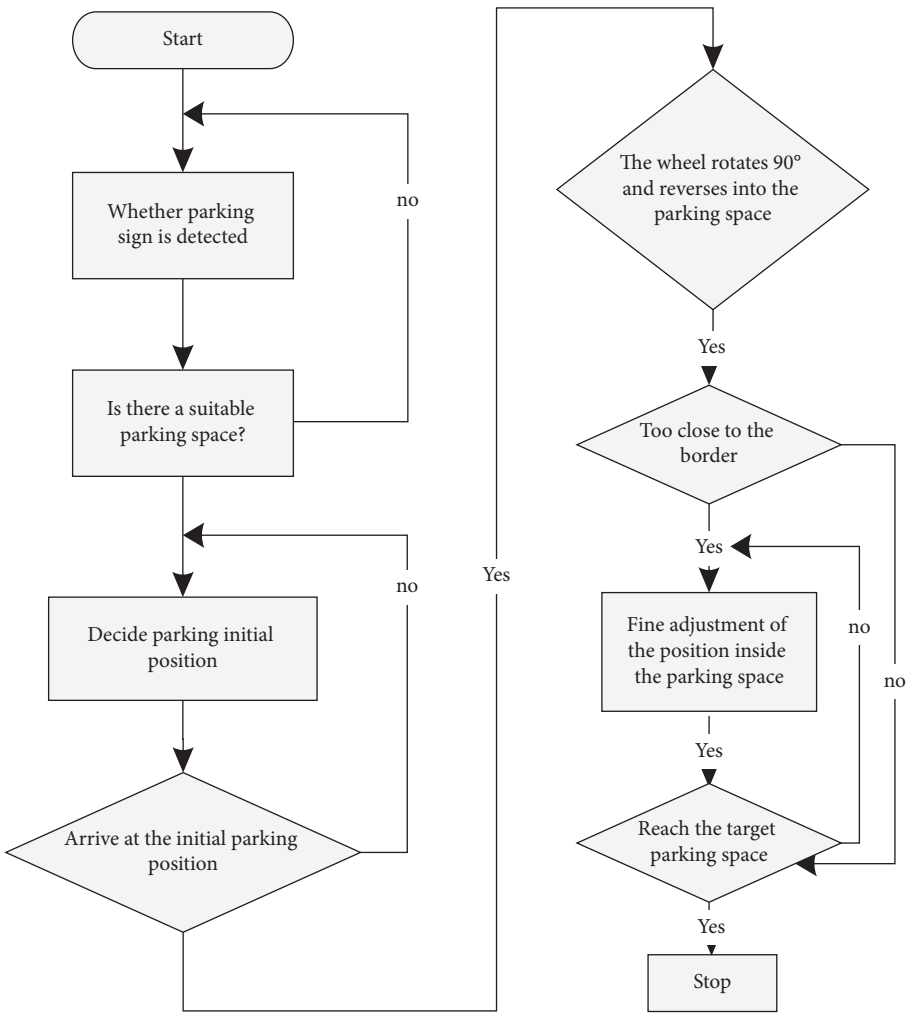


Figure 3: The operation flow of autonomous parking.

Figure 4: Turtlebot3.

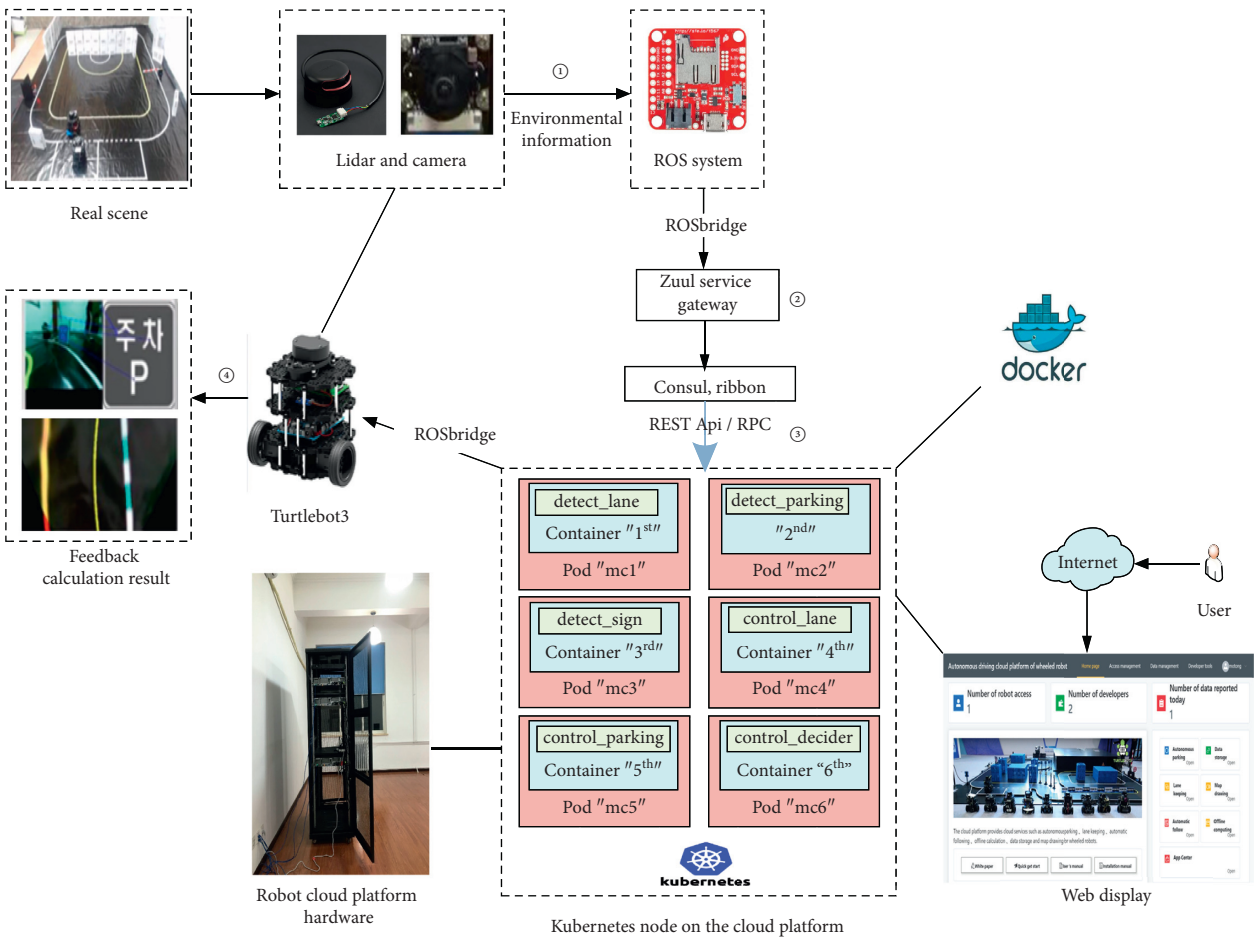


Figure 5: The flow of information between robots and cloud platforms.

experiments can verify that the solution proposed in this article can save more time when calling services.

As can be seen from Figure 8, the total time consumption increases as the number of service calls increases. Compared with the other two schemes, the proposed delivery robot cloud platform based on microservice completes the service call in the shortest time, and the total time consumption gap is also growing. When the number of services is 300, the total time cost gap between the monomer CR system based on ROS and the proposed delivery robot cloud platform based

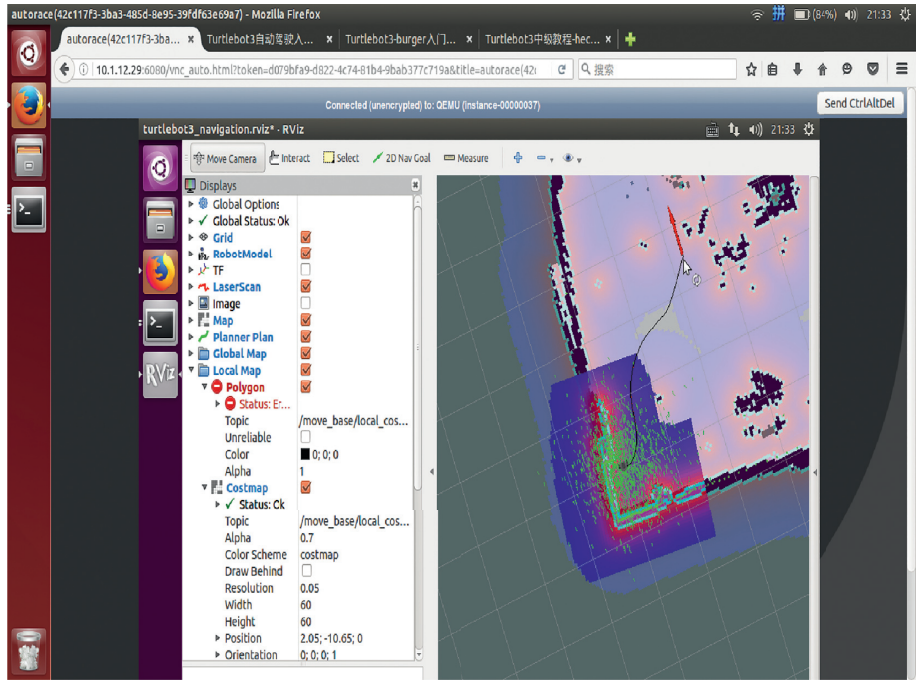
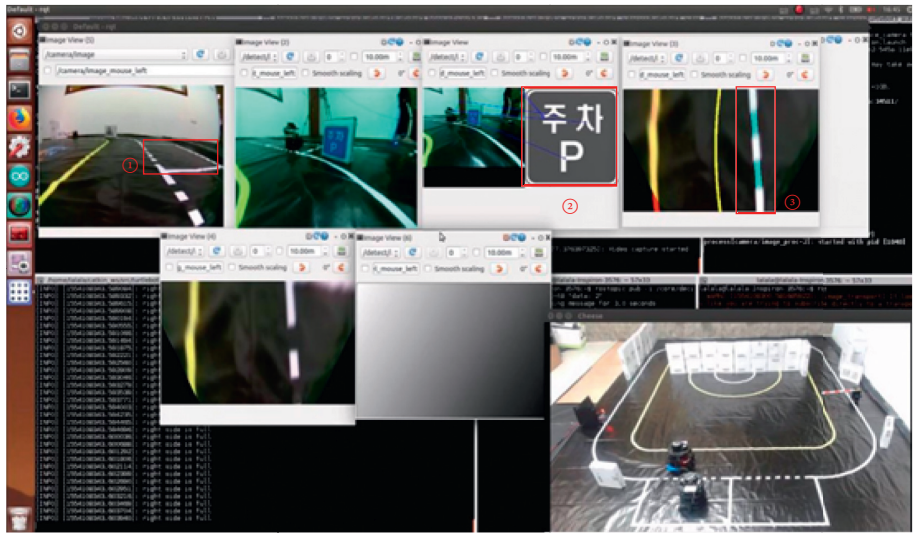


Figure 6: Maps are stored in the cloud.



1) Call the detect_lane service when the stop sign is detected;

2) Call the control_lane and detect_parking service to find the correct parking spot when the dashed lane line is detected;

3) Call the control_decider and control_parking service to park the robot into the parking spot when the parking spot is detected.

Figure 7: Delivery of the prop to a designated parking spot.

Table 1: Comparison of three schemes in detail.

No.	Deployment time	Coupling degree	Scalability	Code maintenance difficulty
Scheme 1	One day	High	Bad	Hard
Scheme 2	Eight hours	Low	Good	General
Scheme 3	Fifty minutes	Very low	Very good	Easy

Note. Scheme 1: The monomer CR system based on ROS; Scheme 2: SOA-based robot cloud platform; Scheme 3: The proposed delivery robot cloud platform based on microservice.

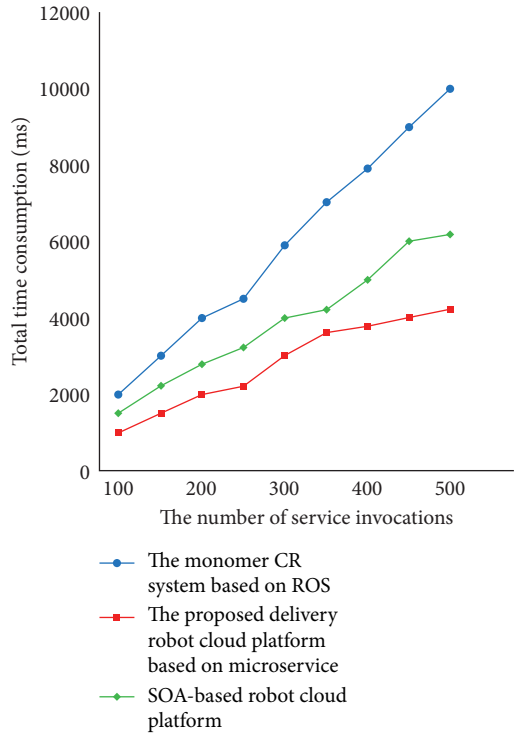


Figure 8: Comparison of the time consumption of the three schemes.

on microservice is close to 3500 ms. From a trend point of view, the gap will be even greater. When running services on a monolithic architecture, the basic needs may be met when few service calls are accessed. When the number of service calls becomes larger, its performance in three aspects is the worst. The results show that the delivery robot cloud platform based on microservice has better performance than the above two frameworks in terms of saving time.

7. Conclusion and Prospect

A delivery robot cloud platform based on microservice is proposed in this paper, and through the delivery scene experiment, it can be concluded that the proposed platform architecture is feasible. Compared to other platforms, this platform enables rapid development and deployment of applications, reducing development time from months to weeks. The platform is currently just a prototype system, and we will continue to conduct in-depth research to improve it further. In the future, we hope to implement the visual composition and orchestration of microservices on the platform, support graphical drag-and-drop development, and make cloud service development easier.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] J. J. Kuffner and S. M. LaValle, "Space-filling trees: a new perspective on incremental search for motion planning," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2199–2206, IEEE, Piscataway, JN, USA, September 2011.

[2] K. Sugiura, Y. Shiga, H. Kawai, T. Misu, and C. Hori, "A cloud robotics approach towards dialogue-oriented robot speech," *Advanced Robotics*, vol. 29, no. 7, p. 449, 2015.

[3] H. Y. Wu, L. Lou, C.-C. Chen, S. Hirche, and K. Kuhnlenz, "Cloud-based networked visual servo control," *IEEE Transactions on Industrial Electronics*, vol. 60, no. 2, p. 554, 2013.

[4] R. Arumugam, V. R. Enti, B. Liu et al., "A cloud computing framework for service robots," in *Proceedings of the 2010 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3084–3089, Anchorage, AK, USA, May 2010.

[5] O. Saha and P. Dasgupta, "A comprehensive survey of recent trends in cloud robotics architectures and applications," *Robotics*, vol. 7, no. 3, p. 47, 2018.

[6] M. Waibel, M. Beetz, J. Civera et al., "RoboEarth," *IEEE Robotics & Automation Magazine*, vol. 18, no. 2, pp. 69–82, 2011.

[7] W. Chen, Y. Yaguchi, K. Naruse, Y. Watanobe, K. Nakamura, and J. Ogawa, "A study of robotic cooperation in cloud robotics: architecture and challenges," *IEEE Access*, vol. 6, pp. 36662–36682, 2018.

[8] R. Doriya, P. Chakraborty, and G. C. Nandi, ""Robot-cloud": a framework to assist heterogeneous low cost robots," in *Proceedings of the International Conference on Communication, Information & Computing Technology*, pp. 1–5, Mumbai, India, October 2012.

[9] R. Doriya, P. Chakraborty, and G. C. Nandi, "Robotic services in cloud computing paradigm," in *Proceedings of*

the International Symposium on Cloud and Services Computing, pp. 80–83, IEEE Computer Society, Mangalore, India, December 2012.

[10] G. Mohanarajah, D. Hunziker, R. D'Andrea, and M. Waibel, "Rapyuta: a cloud robotics platform," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, p. 481, 2015.

[11] M. Tenorth, K. Kamei, S. Satake, T. Miyashita, and N. Hagita, "Building knowledge-enabled cloud robotics applications using the ubiquitous network robot platform," in *Proceedings of the 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5716–5721, Tokyo, Japan, November 2013.

[12] J. Liu, F. Zhou, L. Yin, and Y. Wang, "A novel cloud platform for service robots," *IEEE Access*, vol. 7, pp. 182951–182961, 2019.

[13] C. Berger, B. Nguyen, and O. Benderius, "Containerized development and microservices for self-driving vehicles: experiences & best practices," in *Proceedings of the 2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, pp. 7–12, IEEE, Gothenburg, Sweden, April 2017.

[14] C. Xia, Y. Zhang, L. Wang, S. Coleman, and Y. Liu, "Microservice-based cloud robotics system for intelligent space," *Robotics and Autonomous Systems*, vol. 110, pp. 139–150, 2018.

[15] http://docs.OpenStack.org.

[16] S. Liu, J. Tang, C. Wang, Q. Wang, and J.-L. Gaudiot, "A unified cloud platform for autonomous driving," *Computer*, vol. 50, no. 12, p. 42, 2017.

[17] C. Singh, N. S. Gaba, M. Kaur, and B. Kaur, "Comparison of different CI/CD tools integrated with cloud platform," in *Proceedings of the 2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, pp. 7–12, IEEE, Noida, India, January 2019.