Design and implementation of real-time robot operating system based on freertos

Li Shao[1, 2, 3], Chuanxi Wang[3], Chong Chu[3], Yinan Song[3], Haoyu Hu[3], Yanan Yang[3], Wenyu Fei[3] and Xiaoyang He[1, 2, 3*]

[1]University of Chinese Academy of Sciences, Beijing, 100049, China

[2]Shenyang Institute of Computing Technology Chinese Academy of Sciences, Shenyang, Liaoning Province, 110168, China

[3]Dalian Polytechnic University, Dalian, Liaoning Province, 116034, China

[*]Corresponding author's e-mail: hexy@dlpu.edu.cn

Abstract. With the advancement of artificial intelligence technology, the product of its integration with robots:intelligent robots, is appearing more and more in people's lives and work. Compared to the single function of industrial robots, intelligent robots have the ability to perceive, recognize and decide on the outside world. In order to achieve these functions, data needs to be collected by various types of sensors, and then processed by various algorithms. The introduction of these modules has increased the complexity of the system. Therefore, in order to reduce the complexity between the functional modules, reducing the coupling between modules, and improving the code reuse rate of the functional modules has become an urgent task. This paper proposes an improved FreeRTOS operating system have a driver layer, a middleware layer, and a functional module layer. The experimental results show that the scheme can support the maximum 2000hz module response frequency and the Topic data throughput rate of 2K Bytes per second.

1. Introduction

The International Organization for Standardization defines robots as: "The robot is an automatic, position-controllable, programmable multi-function manipulator with several axes that can handle various materials by means of programmable operation, parts, tools and special equipment to perform various tasks".[1]. Intelligent robots enhance the perception, recognition and decision-making ability of the external environment based on traditional robots [2]. These capabilities are achieved by installing different types of sensors on the robot. A large number of sensors increase the difficulty of robot design and development. Therefore, in order to improve the design and development efficiency of the robot function module and coordinate the effective work of each function module, how to effectively reduce the coupling between the function modules becomes an urgent problem to be solved. The world-renowned ROS (Robot Operating System) is a robotic operating system that allows users to quickly build their own robot platform [3, 4]. The primary design goal of ROS is to increase the code reuse rate in the field of robot development and reduce the coupling and complexity between functional modules. Through the publisher/subscriber communication mechanism, ROS allows any functional module to be mounted on the ROS as a node at any time, and the nodes are well connected, so that the coupling degree of each functional module is greatly reduced. However, ROS is based on the Linux operating system. Linux is a non-real-time operating system and cannot cope with scenarios with high real-time requirements. This article describes a robotic operating system that combines a real-time operating system with a publisher/subscriber communication mechanism in ROS. It can not only solve the real-time demand of intelligent robots, but also improve the function code reuse rate and reduce the coupling degree between functional modules.

2. Real-time operating system selection

Most of the current robot system hardware platforms are based on the ARM core M series MCU, which is characterized by no MMU (memory management unit). Its fast response and strong real-time performance are limited to Flash and SRAM capacity, and most of them are lightweight RTOS. Mainly with multi-task scheduling management, optimize CPU runtime and system resource allocation. Robotic systems typically require processing power for a variety of sensor data, requiring a high degree of real-time and hard interrupt generation, and any upper-level instructions and decisions require the underlying timely response and processing. The ROTS selected by this solution is FreeRTOS. FreeRTOS is an AWS-authorized open source project authorized by MIT. The ecological environment is very stable and will receive more technical support. No matter whether it is scientific research or late products, there is no need to consider the issue of fees. As an RTOS system, FreeRTOS has low complexity and a small number of files. At the same time, FreeRTOS has been ported to many different microprocessors, which is very convenient for use [5-9].

3. Overall system design

The real-time robot operating system proposed in this paper adopts a distributed processing framework. Each functional module is designed as a node and runs under a loosely coupled model. Each module uses PSP (Publish-Subscribe-Pattern) model for communication. The whole system is divided into three layers: device driver layer, middleware layer, and functional module layer. As shown in Figure 1.

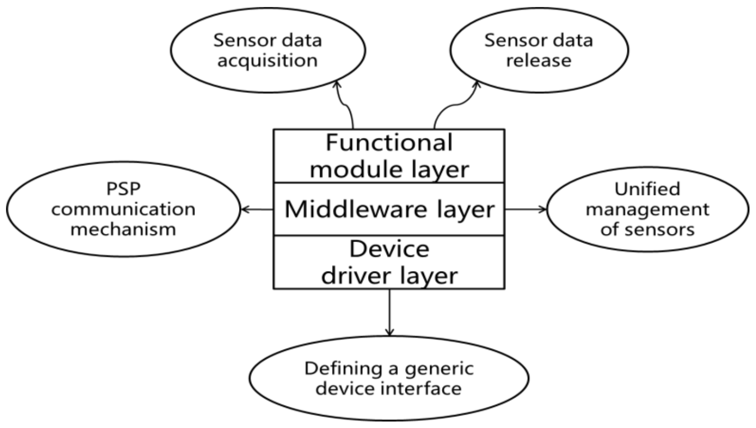

Figure 1. System structure diagram.

The function module layer completes the data acquisition and distribution functions of the sensor device. The middleware layer mainly provides two management services: the first implements the PSP (Publish-Subscribe-Pattern) communication mechanism of the whole system; the second establishes a unified sensor management model. The device driver layer defines a common device operation interface for each device.

3.1. Analysis and design of functional module layers

The design of the functional module layer is to effectively implement code reuse, so the general functions of the functional modules are abstracted into the basic classes. The function module layer includes: an algorithm module, a timer module, a frequency parameter setting module, etc. The timer can periodically arrange the processing functions in the module to acquire data and process the data. The frequency parameter setting module can control the frequency of operation of the submodule.

3.2. Analysis and design of middleware layer

3.2.1. Communication mechanism design
The real-time robot operating system proposed in this paper adopts PSP (Publish-Subscribe-Pattern), which is the publisher/subscriber communication model. Its essence is event monitoring and broadcast model: The subscriber of the message registers a listener with the publisher when needed and registers the topic that they want to subscribe to the message dispatch center. When a publisher has a message to publish, the message to be published is broadcast to the message dispatching center in the form of a topic, and the event notification is triggered to the subscriber through the listener. The dispatching center will uniformly schedule the processing code of the subscriber to the dispatch center.

This scheme implements PSP communication mechanism in the form of cyclic message queue. The topic published by the publisher is defined as a circular message queue, which reads and writes by means of reading and writing pointers. Since each subscriber has its own read pointer, subscribers can control the timing of reading data at their own discretion without affecting other subscribers. This enables a data communication model for multiple publishers of a publisher.

3.2.2. Sensor Management Module Design
The robot system will contain a variety of sensors, such as: motion sensor (IMU), air pressure sensor, GPS, camera, sound sensor, etc. These sensors are the data input sources of the robot. In order to effectively manage these sensors and their data, a sensor management module was designed: SensorManager. The device module Dev is abstracted in the SensorManager. As shown in Figure 2.

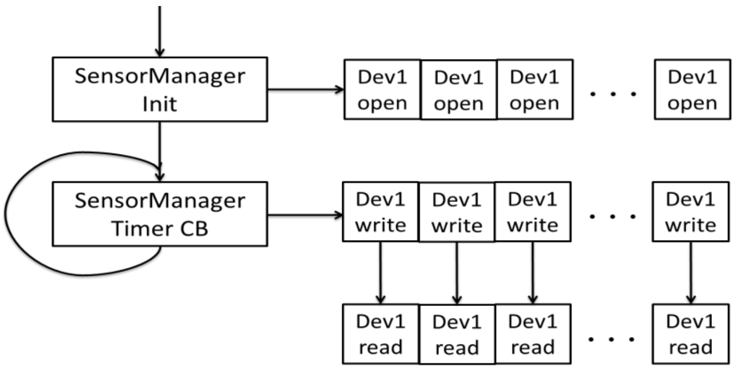

Figure 2. Sensor device call flow chart.

First, all the sensor devices are added to a global sensor device list. When the SensorManager starts and initializes, it traverses each sensor device in the list and calls the open function interface. Each device completes the initialization process of the sensor device by implementing the open interface, and then the timer callback function is periodically scheduled in the SensorManager. In the callback function, iterate through each device registered in the device list, and call each sensor device write function to write the device's timing control instruction to the device, read function to read the sensor number data. Finally, the sensor data received by the TopicManager function is released to the system.

3.2.3. Drive layer analysis and design
The driver layer is a software module responsible for communicating with external hardware devices, and generally includes a hardware abstraction layer (HAL), a board support package (BSP), and a driver. The driver layer is the main channel for communication between the system and external sensors and external control devices. Its main function is to provide the operation interface of the external device and the driver of the device for the upper module. Device drivers can generally be abstracted as: open, close, read, read, write. These operations basically cover all operations of an external device.

4. System implementation and testing

4.1. Test environment
The real-time robot operating system proposed in this paper is verified by STM32F4 [10-12]. The main technical indicators of the system include: frequency pressure, message delay response time, maximum load capacity and stability. The test cases and test results of each performance indicator are described in detail below.

4.2. Test case
According to the scheduling frequency of the system, four test cases are designed.

1) Frequency stress test: Define four modules to run the addition operation at 2000Hz, 1000Hz, 500Hz, 200Hz, and calculate the running frequency of the output module every 3S. At the same time, the error between the actual frequency and the expected frequency is counted.

2) Message delay response test: Add test Topic communication based on test case one. Time stamp and broadcast for each message in the 500Hz module. Then raise the 200Hz module to 500Hz, and test the Topic in it, and receive the receiving timestamp. Compare timestamp errors received and sent, delayed response of statistical messages.

3) Maximum load capacity test: Copying multiple submodules through a submodule template and implementing the receiving operation of the test topic in it. Each module operates at 200Hz. Count the frequency error of each module and the average frequency error of all modules. The error statistics are then output at a frequency of 3S.

4) 24-hour stability test: Start 1000Hz module for Sensor data acquisition; Start the 400Hz module to solve the pose and send the attitude data Topic; Start 200Hz module for attitude data reception. Statistical gesture information output per 3S. These functions perform a 24-hour trial run and count frequency errors, attitude errors, and message delay errors.

4.3. Test Results
1) Results of the first test: In the test, the priority order of the modules is 500Hz>200Hz>1000Hz>2000Hz (The highest priority module is generally set at 500Hz, which can meet the scheduling period of each module in the overall system.). Sampling every 10 minutes, sampling a total of six times, the results are shown in Table 1:

Table 1. Frequency pressure data.

Test frequency	1 sampling	2 sampling	3 sampling	4sampling	5 sampling	Average sampling
2000Hz	1989	1926	1975	1894	1988	1907
1000Hz	978	986	976	984	977	988
500Hz	498	499	491	497	497	495
200Hz	198	199	195	197	199	196

It can be seen from the experimental data that the real-time operating system running frequency is stable at 200-1000 Hz. When the frequency domain is running to 2000 Hz, the scheduling performance loss has begun to increase gradually. Therefore 2000Hz is basically the maximum scheduling frequency of the real-time system. If you want to implement a high frequency module, you need to use a hardware timer to complete. For high frequency modules, a hardware timer is used to complete.

2) Results of the second test the size of the message is 2048B, 1024B, 256B, 64B, and a total of six samples, the results are shown in Table 2.

Table 2. Message delay response data.

Message data volume	1 sampling average	2 sampling average	3 sampling average	4sampling average	5 sampling average	Average sampling
2048 B	30us	29us	31us	30us	32us	31us
1024 B	29us	29us	30us	29us	29us	30us
256 B	30us	29su	32us	28us	29us	29us
64B	30us	29us	29us	31us	32us	31us

During the experiment, I found that the amount of Topic data will affect the frequency of its data update. Because Topic is a circular queue buffer, its buffer size is fixed. Therefore, if the amount of Topic data is too large, the number of buffered cache data is too low, so that too much data cannot be cached. If the subscriber module does not update the data in time, it will result in data loss. This real-time operating system can support Topic data cache of 2048B per second.

3) Results of the third test: In the test, Tested the load capacity of 100, 80, 50, 20 submodules respectively. All modules have a priority of normal. The average operating frequency of each type of module is sampled every 10 minutes, and a total of 6 records are recorded. The results are shown in Table 3.

Table 3. Maximum load capacity test data.

Number of load modules	1 sampling average	2 sampling average	3 sampling average	4sampling average	5 sampling average	Average sampling
100	193	192	194	193	194	193
80	197	196	193	195	195	196
50	199	198	199	198	199	199
20	200	200	200	200	200	200

It can be seen from the experimental data that the delay of the message will increase as the number of system modules increases. As the number of modules increases, the period in which a message needs to be distributed becomes longer. In robot systems, some messages require real-time broadcast and response; some message delays have little effect on the system. Therefore, when designing the module, it is necessary to divide the purpose of the topic, and set the upper limit of the subscriber according to the demand for the response.

4) Results of the fourth test: In the test, the system status indicators are saved every 1 minute. The indicators include: the actual running frequency of the module, the average delay time and the maximum delay time of the Topic data transmission-reception. The system has accumulated 24 hours of operation (1440 points). The results are shown in Table 4.

Table 4. 24-hour stability test data.

Sampling point	1000Hz module frequency	400Hz module frequency	200Hz module frequency	Topic average delay	Topic maximum delay
1	1000	400	198	30us	35us
2	1000	400	199	28us	35us
3	999	400	195	31us	36us
4	999	400	194	29us	34us
5	1000	400	199	31us	35us
6	1001	400	198	30us	34us
7	999	400	197	29us	33us
8	1000	400	196	28us	35us
...					
1440	1000	400	199	32us	35us

It can be seen from the experimental data that the stability of the real-time operating system is good, and the experimental data of 1440 minutes has no large fluctuations.

5. Conclusion

This paper proposes a real-time robot operating system based on FreeRTOS. The system leverages the ROS publisher/subscriber communication mechanism to combine the real-time nature of FreeRTOS. The whole system is divided into three layers: drive layer, middleware layer and functional module layer. The middleware layer is the core of the whole system, which can uniformly manage the sensor and PSP communication mechanism. The sensor management module is responsible for uniformly scheduling the interface functions of each sensor and controlling the access timing of the sensor. The user can access the system to work by simply inheriting the driver device base class and adding it to the driver device list. The PSP communication module uses Topic as the message hub of the entire middleware, responsible for the establishment of Topic and the distribution of messages. The message data area is stored by using the message data wake-up queue, the subscription list information is saved by using the subscriber list, and the message distribution is completed by updating the subscriber list when the message is updated, thereby speeding up the message forwarding process.

Through theoretical analysis and experimental data, the system can better meet the user's requirements for the robot system in real-time and load capacity. In the subsequent research process, some common modules will be developed by themselves, such as log module, IMU module, SLAM module, etc. These modules can better verify the scalability of the system.

References

[1] International Federation of Robotics 2018 Industrial robotics standardization.http://www.ifr.org/news/ifr-press-release/iso-roboticsstandardisation-35/.
[2] TianMiao Wang, Yong Tao, Hui Liu 2018 Current researches and future development trend of intelligent robot: a review [J]. International Journal of Automation & Computing, 9:1-22.
[3] Quigley M, Gerkey B, Conley K, Faust J, Foote T, Leibs J, Berger E, Wheeler R and Ng A 2009 ROS: An open-source Robot Operating System. ICRA Workshop on Open Source Software. 3: 1-6.
[4] Cun-xu Hu 2018 *ROS robot development practice*. China Machine Press, Beijing. (In Chinese)
[5] Rui-jie Chen. (2016)Overview of the Development of Real Time Operating System RTOS [J].Telecom Power Technology. 4:199-200. (In Chinese)
[6] Zhong-kai Zuo, Jun Liu and Yang Zhang 2018 *FreeRTOS source code and application development*. Beihang University Press, BeiJing. (In Chinese)
[7] Cedeno W and Laplante P 2007 An Overview of Real-time Operating Systems. Journal of the Association for Laboratory Automation, 12(1), 40–45.
[8] Fei Guan, Long Peng, Luc Perneel and Martin Timmerman 2016 Open source FreeRTOS as a case study in real-time operating system evolution [J].Journal of Systems and Software, 118:19-35.
[9] Richard Barry 2019 Mastering the FreeRTOS Real Time Kernel. Http: // www.FreeRTOS.org
[10] Zhao Bo 2019 Temperature measurement system design based on STM32 and FreeRTOS. Electronic Technology and Software Engineering. Electronic Technology&Software Engineering, 3:68-69. (In Chinese)
[11] Zhang Shun,Yan Hong-zhi,Han Feng-lin and Li peng.2018 Design of Go Robot Controller Based on ARM and FreeRTOS, Manufacturing Automation, 40(1):28-32.
[12] Ping Qian, Yingzhen,Zhang and Yu Li 2015 Design of voice control system for smart home based on STM32 [J].Applied Mechanics and Materials, 734:369-374. (In Chinese)