Design of Serial Multi-Motor Communication Programme Based on CAN Bus Master-Slave Architecture

Wei Jia Xu[1], Tao Zheng[2,]*

[1]Geely University of China; China; xu18280141482@163.com

[2]*Geely University of China; China; 15884408147@163.com

Abstract. Motor control plays a crucial role in daily life, but challenges often arise in terms of precise control and scalability. By modularizing motor control using a CAN bus master-slave architecture and a built-in PID control servo system, modular expansion allows for the selection of the number of motors to be controlled. Through the host, each motor's state can be precisely controlled, improving the control accuracy and scalability of multi-motor systems. This paper presents the system architecture and code implementation of the serial multi-motor Communication programme based on the CAN bus master-slave architecture, which holds significant practical value in industrial machine applications.

CCS Concepts: Computer systems organization → Embedded and cyber-physical systems → Sensor networks

Keywords: Communication protocol design; Serial communication mechanism; Network topology; Motor control system; CAN communication

1.Introduction

Against the broader backdrop of the intelligent transformation of the global manufacturing industry and the accelerated advancement of "Industry 4.0," the demand for multi-motor cooperative control systems in strategic sectors such as high-end equipment manufacturing, new energy vehicles, and intelligent robotics has experienced explosive growth[1-3]. By 2020, the global installed base of industrial robots had exceeded 3 million units. The Asian region not only exhibited the fastest growth but also accounted for the largest share, comprising 69% of

* Corresponding author: l15884408147@163.com

the global installed capacity[4]. Traditional multi-motor control programmes predominantly employ centralized PLC or dedicated controller architectures, which present bottlenecks including complex wiring, high expansion costs, and weak anti-jamming capabilities. The core challenge lies in achieving high-precision synchronization and dynamic coordination among multiple motors. Moreover, traditional static IP bonding and star network architectures have struggled to meet the demands of rapid deployment[5]. In recent years, fieldbus-based distributed control technologies (e.g., CAN, Ether CAT) have enhanced system flexibility to some extent; however, existing solutions still exhibit significant limitations in dynamic node management, communication delay compensation, and fault self-healing capabilities[6, 7].

Based on existing research, the CAN bus master-slave architecture has gradually become the mainstream approach for distributed control of multiple motors due to its high reliability, real-time performance, and anti-interference capabilities [8]. In this paper, we propose a serial multi-motor communication approach based on the CAN bus master-slave architecture, achieving accurate and cooperative control of multiple motors through modular hardware design and a master-slave communication protocol. Experimental results demonstrate that the system exhibits excellent communication stability in both complex environments and strong interference scenarios, providing a high-precision and highly scalable solution for industrial robots, electric vehicle drives, and other applications.

2.Overall design of system

The serial multi-motor communication scheme based on the CAN bus master-slave architecture consists of two parts: the host transmitter and the slave controller. The main advantages of this scheme include modular cascade scalability, a master-slave control network for serial nodes, and an automatic dynamic node coding protocol. The main advantages of the serial multi-motor communication solution based on the CAN bus master-slave architecture include modular cascade scalability, a serial node master-slave control network, and an automatic dynamic node coding protocol. All slaves are isomorphic and modular, offering greater flexibility than static IP bindings for individual nodes. Based on the CAN bus master-slave architecture, the host sends data to the slave, which then receives the data and uses the PID control algorithm to

achieve accurate servo speed control of the motor. The operator can select a slave and configure its data through the master.

The control relationship between the master and slave is illustrated in Figure 1.

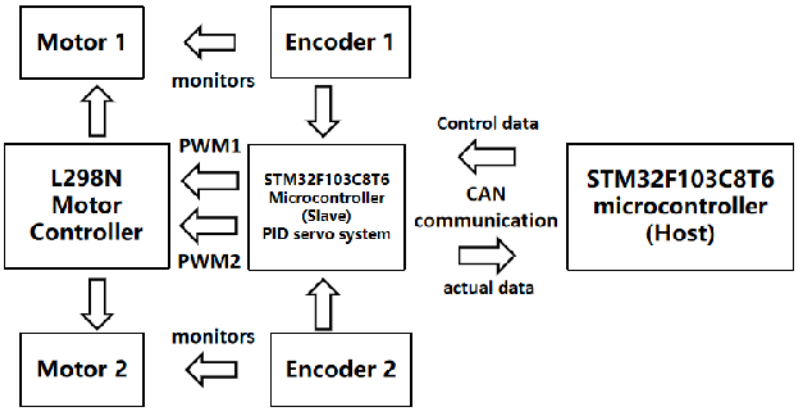

Figure 1. Master and slave control relationship diagram

3.CAN bus-based master-slave serial architecture design

In the serial multi-motor Communication programme based on the CAN bus master-slave architecture, the design of the master-slave serial architecture and communication protocol is crucial. A well-designed communication protocol ensures accurate and efficient information exchange between the master and slaves, thereby enabling precise control of multiple motors. Such a protocol not only fully leverages the technical advantages of the CAN bus but also meets system requirements for dynamics, scalability, and reliability, thereby providing a solid foundation for the stable operation of the entire multi-motor control system [8].

3.1. Technical characteristics of the CAN bus

The CAN bus exhibits high reliability, with robust anti-interference capabilities that resist electromagnetic interference in industrial environments to ensure stable signal transmission. Its error self-testing mechanism can promptly detect and correct transmission errors to guarantee data accuracy. Additionally, the automatic fault node disengagement function effectively isolates faulty nodes, preventing their impact on the operation of the entire system. Together, these features ensure the stability of the CAN bus and the reliability of data transmission under complex working conditions, providing a solid foundation for the precise control of multiple motors [9, 10].

3.2. Master-Slave String Architecture Design

The CAN bus supports a multi-master, multi-slave communication mode, making it ideal for constructing a multi-motor cascade control system within a master-slave architecture. A system can contain one or more masters and multiple slaves. Masters can flexibly communicate with each slave to send control commands and receive status information [11-13].

3.2.1. Master-Slave Node Functional Classification

(1) Host Function

The host is equipped with a human-computer interaction interface, allowing the operator to input control data. The host is responsible for parsing these commands, determining the target slaves to be controlled, and specifying the motor operating parameters (which can be point-to-point data frames or cluster packets). According to the operator's commands, the host accurately distributes the data to each slave.

The master periodically sends data requests to the slaves via the CAN bus in sequential order, as shown in Figure 2. The slave receiving the data request responds with real-time motor status data, such as speed, current, and temperature. The master stores and analyzes this data. The host also stores and analyzes these data to monitor the motor's operational status in real time, ensuring the stable operation of the entire system.

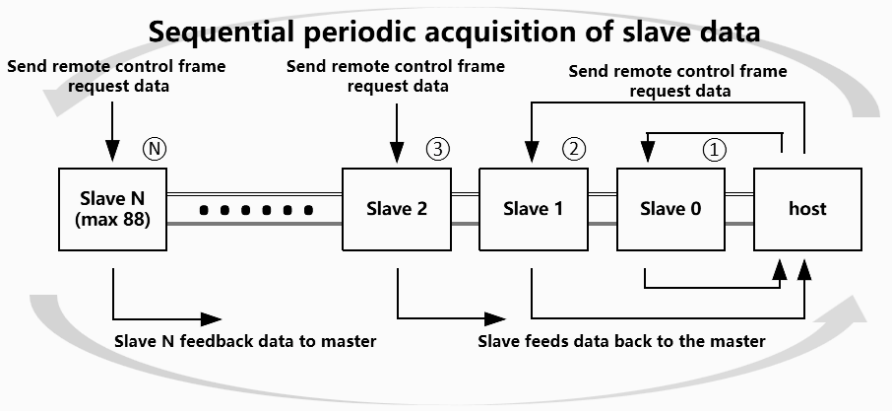

Figure 2. The master acquires slave data in a sequential periodic manner

When the host detects abnormal data returned by a slave, it can promptly identify a potential fault in the motor or the slave controller. The master then sends a stop command to the faulty slave and simultaneously issues an alert to notify the operator for timely maintenance and intervention.

(2) Slave Function

The slave receives data from the host via the CAN bus, parses the content, and controls motor operation accordingly.

The slave is connected to the motor encoder, current sensor, temperature sensor, and other monitoring devices to collect real-time data on motor speed, current, temperature, and other status parameters. This information is encoded using a predefined data format and transmitted to the host via the CAN bus upon receiving a data request.

The slave also possesses basic local control capabilities. For instance, if the motor current or temperature exceeds a predefined threshold, it immediately initiates protective actions, including cutting off the motor's power supply, issuing an alarm, and transmitting overload or fault information to the host to prevent damage to the motor or the slave unit.

3.2.2. Master-Slave Architecture Tandem Connection Design

Both the host and the slaves are equipped with the TJA1050 chip as the CAN bus transceiver and are interconnected via the two signal lines (CAN_H and CAN_L) of the CAN bus. The host's CAN bus interface is connected to that of Slave 0. The slaves are connected in a daisy-chain configuration, as shown in *Figure 3*: Slave 0 is connected to Slave 1, Slave 1 to Slave 2, and so forth. This configuration facilitates easy expansion of slave nodes and enables cascade control of multiple motors.

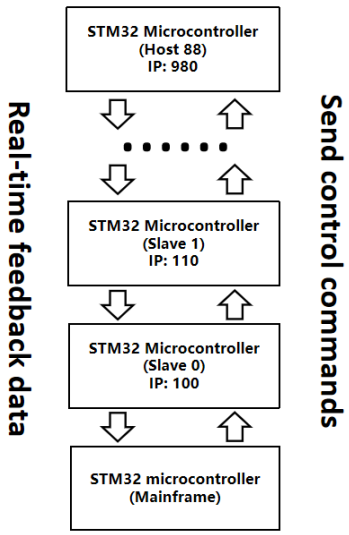

Figure 3. Design of Serial Connections in Master-Slave Architecture

The host provides a stable power supply to the slaves, while each slave forwards power to the subsequent slave through dedicated inter-slave power interfaces; this power delivery

mechanism ensures that each slave receives sufficient current to meet the operational demands of the motor, and, as the slaves do not carry onboard batteries, they can be miniaturized to save space and enhance portability.

3.3. Dynamic Node Coding Protocol

To enable flexible system expansion and automatic identification of slave nodes—while avoiding the inconvenience and potential errors associated with manual IP address configuration [14], this paper proposes a dynamic node addressing protocol. The core of this protocol is that, upon connection to the CAN bus, each slave is automatically assigned a unique IP address by the host, enabling precise management and communication.

3.3.1. Principles of code assignment

(1) Principle of Uniqueness

In the CAN bus network, each slave node must be assigned a unique IP address to ensure that the host can accurately identify and communicate with each slave. This requirement is fundamental to the dynamic node addressing protocol and is enforced through a rigorous address allocation and conflict detection mechanism.

(2) Principle of Continuity

To simplify management and improve address space utilization, IP addresses should be allocated as continuously as possible. The host assigns addresses to slave nodes sequentially based on their connection order, thereby minimizing address fragmentation and reducing waste.

3.3.2. Coding allocation process

When a slave node is powered on for the first time (i.e., without a stored IP address), it first verifies the communication status of the CAN bus to ensure connectivity with the network. It then checks its flash memory for an existing IP address. If none is found, it sends a dedicated broadcast request frame to the host, requesting assignment of a unique IP address. The purpose of this request is to notify the host that a new slave node has joined the network and to initiate the dynamic IP assignment process. The format and content of the broadcast request frame must conform to a predefined communication protocol, which includes the slave's unique hardware identifier (UID) to ensure correct identification and processing by the host.

The master continuously monitors communication on the CAN bus, and upon receiving a broadcast request frame from a slave, it assigns an available IP address based on predefined rules and the current number of slave nodes in the network. The master then generates a response frame containing the assigned IP address and the slave's unique hardware identifier (UID), and broadcasts it over the CAN bus. Simultaneously, the master sends the same information directly to the requesting slave for confirmation. The master also records the assigned IP address and corresponding UID in its internal node management table to support future communication and system maintenance.

Under normal conditions, all slaves in a serial network receive the response frame broadcast from the host. However, slaves that have already been assigned an IP address or have received an acknowledgement frame will disable their response frame reception channel, thereby filtering out subsequent response frame broadcasts and ensuring that their IP addresses are not overwritten. Slaves that have not yet been assigned an IP address will process the broadcast response frame by verifying whether the hardware identifier (UID) in the frame matches their own, to confirm that the frame is intended for them. If the UID matches, the slave extracts the assigned IP address from the frame and stores it in its internal Flash memory to ensure data retention in the event of a power failure, and then disables its response frame reception channel.

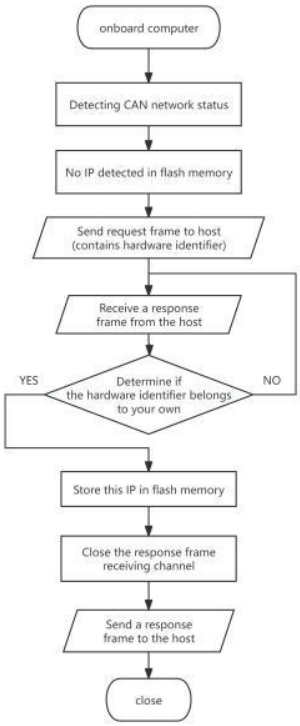

Figure 4. Dynamic Node Encoding Process

After storing the assigned IP address, the slave sends an acknowledgement frame to the host, indicating successful receipt and storage of the IP address. At this stage, the dynamic IP address assignment process for the slave is considered complete. This process is illustrated in Figure 4.

3.3.3. Power back up from the machine

In the event of an unexpected or intentional system power failure, upon the next power-up, the slave reads its stored IP address from flash memory and sends its hardware identifier along with this IP to the host. The host then compares this information against its node management table. If the IP address in the node management table matches the one associated with the hardware identifier, the host replies with a confirmation frame, allowing the slave to continue using the current IP address. If the IP address is already assigned to another node or exceeds the allowable range for the current network, the slave will be assigned a new IP address following the predefined rules.

Slaves may also actively clear their stored IP addresses using hardware methods. This process is illustrated in Figure 5.

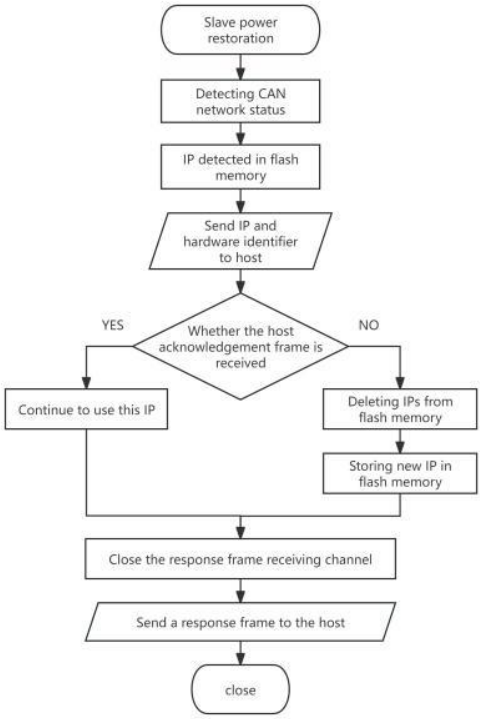

Figure 5. Docking process for re-powering from the machine

3.3.4. Code Management and Troubleshooting

If a slave goes offline due to failure or human intervention, the host must detect this promptly.

Two scenarios arise when a slave goes offline: (1) the slave fails independently, which does not affect the overall operation of the master-slave tandem network; (2) the slave is taken offline by human intervention, is completely damaged, or the connecting cable between slaves is broken. In the latter cases, the master-slave tandem network is compromised, preventing all subsequent slaves, including the affected one, from communicating with the master.

If the above situation (1) occurs, during the sequential periodic acquisition of slave data, the faulty slave will fail to respond when the host requests data. The host will detect a reply timeout and, upon confirming this, will identify the slave controller as faulty. Consequently, the host will skip the faulty slave and simultaneously issue an alarm to notify the operator to perform timely maintenance, as shown in Figure 6.

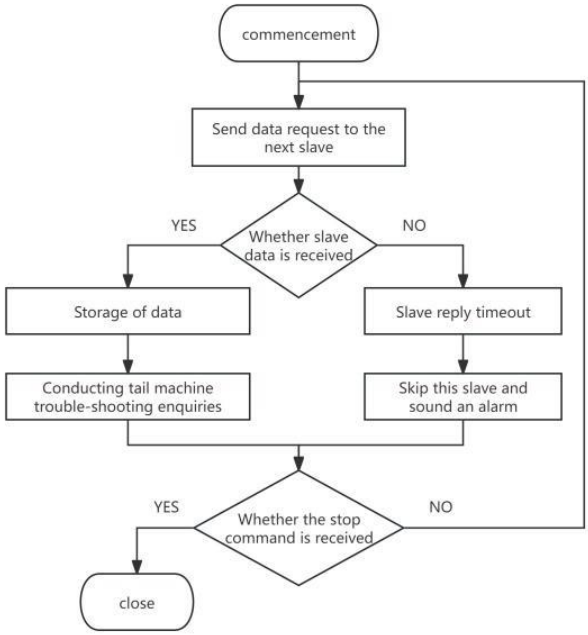

Figure 6. Slave Individual Troubleshooting Process

If the above situation (2) occurs, to quickly determine whether the master-slave serial network is damaged, the master will individually query the end slave during intervals between regular data requests. If the network is damaged, the host will fail to receive a reply from the end slave. After two unsuccessful fault query rounds, the host will terminate all slave tasks and initiate a fault troubleshooting procedure. The host employs a dichotomous approach to fault query the slaves sequentially to quickly identify the faulty slave and continuously alert the operator until intervention. Once the fault is resolved and the operator presses the reset button, the host will

recheck the master-slave serial network. If the network is restored, the host will notify the slaves to resume their tasks, as shown in Figure 7.

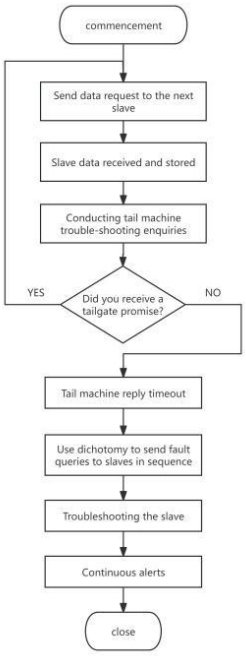

Figure 7. Master-Slave Network Compromise Process

3.3.5. IP address conflict detection and resolution

Despite the principle of uniqueness and a strict allocation mechanism, IP address conflicts may still arise in certain cases. These include when two or more slaves power up simultaneously, causing errors due to the overlap of broadcast request frames, or when a slave's memory failure leads to loss of its IP address, resulting in a conflict upon re-requesting allocation.

When multiple slaves are powered on simultaneously, CAN bus arbitration relies on bit-by-bit priority comparison of the identifier (IP), not the content of the data segment. As shown in Figure 8. Since all slaves send the same IP in the broadcast request frame, the CAN bus cannot distinguish priority via the arbitration mechanism. As a result, slaves will continue sending request frames, and after the arbitration segment, they will send data segments. During transmission, the slaves monitor the bus level through Bit Monitoring, comparing the transmitted bit with the bus level in real time. If an inconsistency is detected in the data segment (e.g., an implicit "1" is transmitted but an explicit "0" is received), the slave triggers a Bit Error, halting transmission of the current frame and transmitting an error frame instead. All nodes on the bus (including the host) will then listen to the error frame. The master will immediately stop receiving data and discard the previously received information.

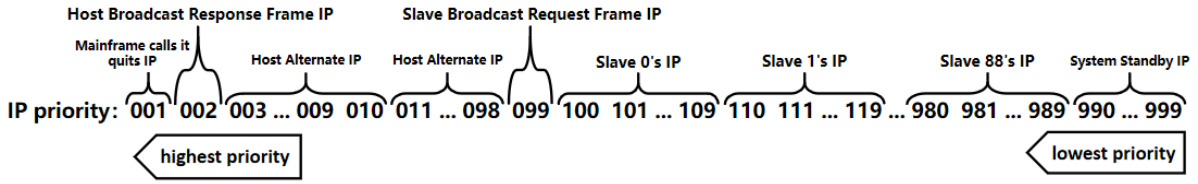

Figure 8. CAN bus IP priority

Simultaneously, the newly accessed slave will attempt to send a request frame again. If it still detects a bus level error, this indicates that multiple slaves have been accessed simultaneously. At this point, all newly accessed slaves will send a special error frame to the host, which contains the same identifier and identical data content (including the same CRC check value). The host will treat the bus data as a single valid transmission, receive the message normally, and continue to alert the operator upon detecting the special error frame until the operator intervenes. Meanwhile, slaves that have not been assigned IPs will trigger a red-light alarm.

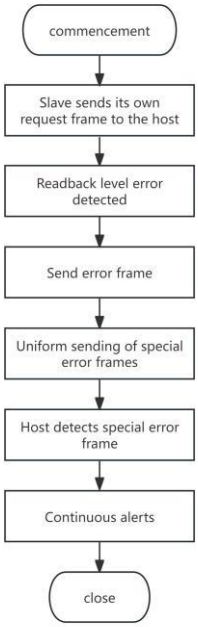

Figure 9. IP Conflict Handling Process

Through the design and implementation of the aforementioned dynamic node coding protocol, the serial multi-motor Communication programme based on the CAN bus master-slave architecture can flexibly manage slave joining, leaving, and failures, ensuring each slave has a unique IP address. This enables efficient and reliable communication and control. The dynamic coding method enhances the system's scalability and maintainability, providing a solid foundation for the cooperative control of multiple motors, as shown in Figure 9.

3.4. Communication protocol design

The IP address assignment protocol is illustrated in *Figure 10*.

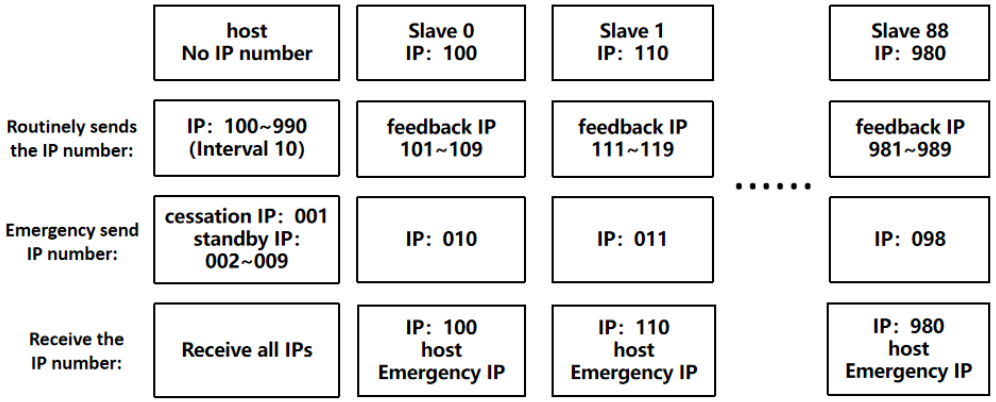

Figure 10. Communication Protocol IP Number Assignment Table

3.4.1. Host Send Protocol

The host's regular send IP addresses range from 100 to 980 (assigned to slaves), while emergency send IPs range from 001 to 009 and serve special purposes. Regular send IPs are used for point-to-point communication, whereas emergency send IPs are broadcast to all slaves. When the host sends data using a regular IP, slaves receive only the data corresponding to their assigned IP and perform the appropriate operations.

The host transmits two frame formats: data frames and remote-control frames. Both use the standard CAN bus frame format, which includes arbitration, control, data, CRC, and ACK segments.

The master sends three types of frame combinations:

1. regular send IP with data frame, for normal data transmission to a single slave.
2. regular send IP with remote control frame, for sending data requests to slaves.
3. emergency send IP with data frame, for broadcasting emergency commands to all slaves.

3.4.2. Slave Receive and Response Protocol

The slave continuously monitors the data flow on the bus via the CAN bus interface. When the host sends a receivable frame, the slave first checks the frame type. If it is a data frame, the slave receives and processes the data regardless of whether the host IP is a regular send IP or an emergency send IP. This is because the underlying control logic of the slave remains consistent regardless of the network layer protocol used. If the frame is a control frame, the

slave encodes the motor's real-time status information—such as rotational speed, current, and temperature—according to a preset data format and transmits this feedback to the host via the CAN bus.

Upon receiving control data from the master and completing data verification, the slave immediately adjusts the motor's operating status based on the data content.

3.4.3. Slave Request and Feedback Protocol

The slave proactively sends data in the following scenarios: during power-up when requesting an IP address, upon receiving a data request from the host, in case of slave failure, motor overload, overheating, and other related conditions.

When first powered on, the slave sends a broadcast request frame with IP 099. Upon receiving this frame, the host reads the hardware identifier and records it in the node management table, then assigns an IP address to the slave based on the current number of slaves in the network. If multiple devices power up simultaneously, they send a special error frame with IP 099 containing identical data to the host.

The slave's transmit IPs correspond to the last nine digits of its own IP address, ranging from xx1 to xx9. Additionally, the slave has an emergency transmit IP, which delivers information—primarily fault data—to the host with higher priority. This emergency transmit IP is calculated by dividing the slave's own IP address by 10. For example, if the slave's IP is 660, its transmit IPs range from 661 to 669, and its emergency transmit IP is 066.

3.4.4. Host broadcast Communication protocol

When the host uses emergency send IPs, it broadcasts messages to all slaves. IP 001 is used to send a stop broadcast command to all slaves and motors, instructing them to immediately halt operations—such as in cases of master-slave serial network failure or emergency stops. IP 002 is used to broadcast a response frame; all slaves in the serial network receive this frame, and if the response matches, the slave extracts the IP address from the frame and stores it in internal flash memory. IPs 003 to 009 serve as alternate IPs, which can be customized by the operator—for example, IP 003 may represent a full-speed motor rotation command.

4.Computer programming

The program is developed in C, using Keil uVision5 as the design software. The flowchart of the main program is as follows.

4.1. Host computer program

(1) Initialization Phase

Configure the system clock, CAN bus baud rate (500 kbps), GPIO ports (key/display interface), and other settings according to system requirements.

(2) Master Loop Logic

Upon receiving slave messages, process them according to the preset programme. When data transmission is required, read the user-entered parameter settings and motor selection, construct CAN data frames, and send them. Periodically send status query requests to the slaves, parse the feedback, and update the interactive interface display. In case of faults, perform fault detection and emergency handling, such as timeout retransmission or node failure management. The main flow of the host is illustrated in Figure 11.

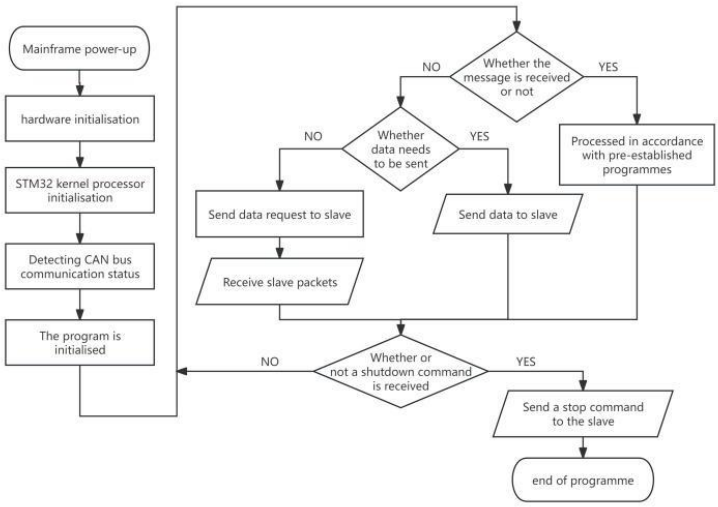

Figure 11. Host Main Flowchart

4.2. Slave programme

(1) Initialization Stage

Read the IP address stored in Flash memory (applicable when powering up in IP application mode for the first time). Configure the TIM timer for PWM output at a frequency of 20 kHz, set up ADC sampling, and initialize the encoder.

(2) Main Loop Logic

Receive incoming data and determine whether it is a data frame or a remote-control frame. If it is a data frame, read the frame data, update the target value, and control the motor using a PID algorithm. If it is a remote-control frame, package the most recent motor data and send it to the host computer. Subsequently, monitor the motor and collect data. The main flow of the slave is illustrated in Figure 12.

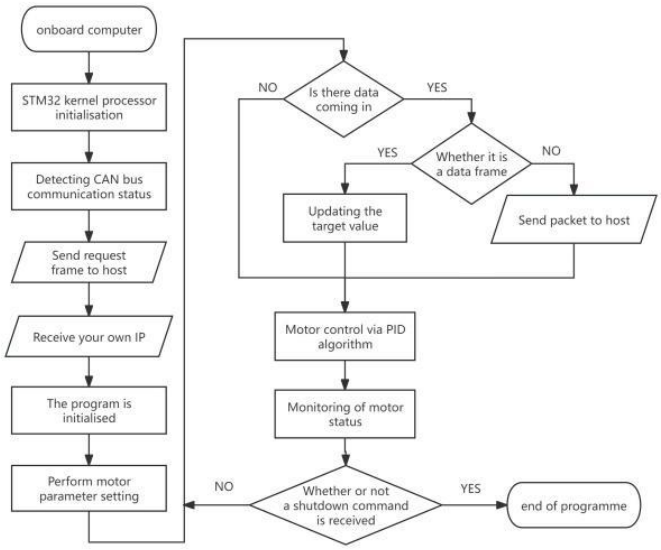

Figure 12. Slave Master Flowchart

5.Practical validation of the programme

5.1. Theoretical basis for selection of experimental hardware

STM32F103C8T6 and TJA1050 were chosen as the communication carriers for this experiment. The STM32F103C8T6 offers powerful processing capabilities and low-power design, making it ideal for multi-motor control systems that require long-term, stable operation. Its built-in low-noise design, combined with the differential transmission technology of the CAN bus, effectively mitigates electromagnetic interference in industrial environments. The TJA1050, a transceiver designed specifically for CAN bus communication, uses differential signal transmission to resist common-mode interference. It features a wide operating voltage range and robust fault detection and self-recovery functions, ensuring stable data transmission in harsh environments.

5.2. Experimental environment setup

The experimental setup consists of a PC-based computer, four STM32F103C8T6 minimum system boards, four TJA1050 communication modules, a DAP downloader, four red LEDs, and several buttons. In this setup, three slaves and one master were powered through the DAP downloader, as illustrated in Figure 13.

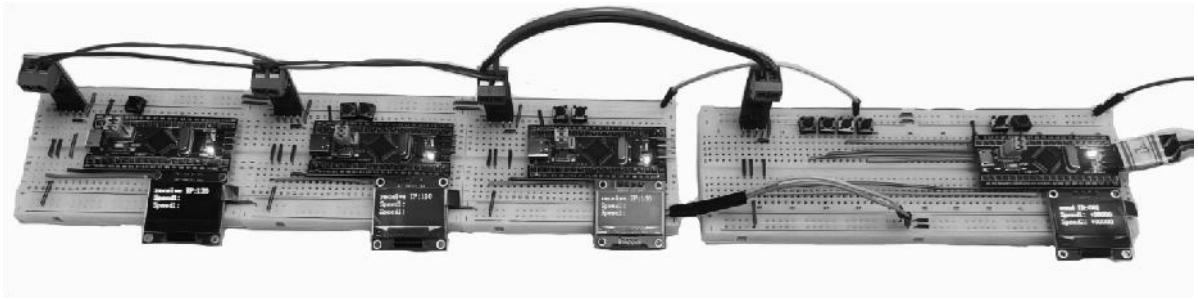

Figure 13. Physical drawing of system construction

5.3. Communication Configuration

The basic CAN communication configuration parameters are listed in Table 1 and Table 2.

Table 1. Basic configuration parameters for CAN communication

Dataset	baud	APB1	BRP	TS1	TS2	SJW	inaccuracy
host	500kbps	36MHz	4	5	2	1	≤0.33%
Slave	500kbps	36MHz	4	5	2	1	≤0.33%

Table 2. CAN communication mode selection

Dataset	Bit Width Setting	Filter Settings	Mode Setting	Related Settings
host	16-bit wide	all pass state	Masking Mode	FIFO0
Slave	16-bit wide	List Status	list mode	FIFO0

5.4. Programme validation

In the experimental validation, we will assess the performance of the CAN bus-based serial multi-motor Communication programme from several perspectives. The effectiveness of the coding programme is crucial to ensuring error-free communication between the master and slave, as well as accurate identification and assignment of IP addresses. We will evaluate the accuracy and real-time performance of data transmission to ensure efficient and stable communication during simultaneous multi-motor operation. Additionally, we will test the troubleshooting solutions, including how to address equipment or network faults promptly to ensure system reliability and stability.

5.4.1. Coding programme validation

The slaves were sequentially connected to the CAN bus network: first, the initial slave was connected as shown in Figure 14; then the second slave was powered on without network connection as shown in Figure 15; finally, the second slave was connected to the network as shown in Figure 16.

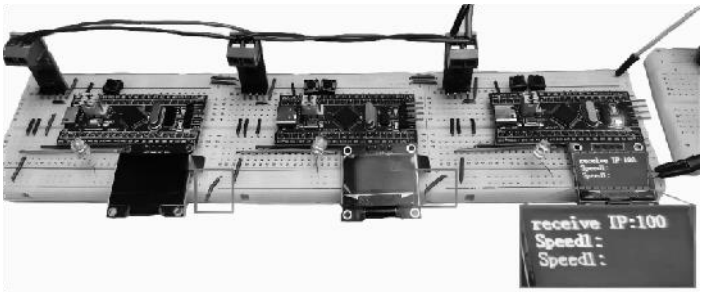

Figure 14. Access to the first slave

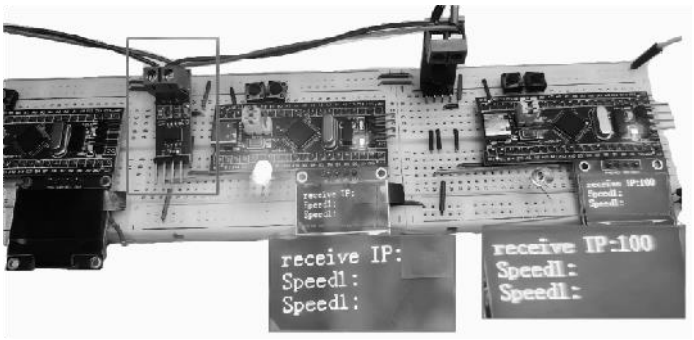

Figure 15. Access to the second slave but not to the CAN bus

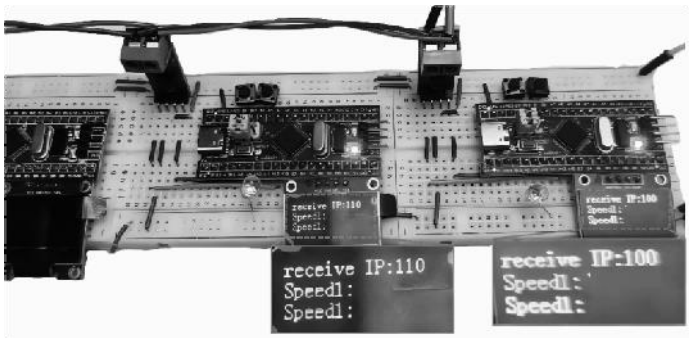

Figure 16. Access to the second slave and to the CAN bus

5.4.2. Transmission Data Validation

Data is sent to the slave via the master to verify whether the slave receives both positive and negative data, as shown in Figure 17.

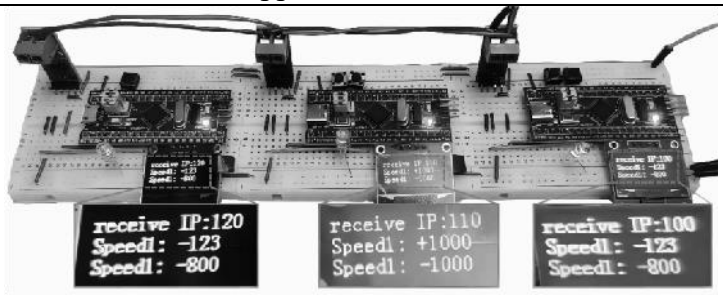

Figure 17. Master transfers data to three slaves

5.4.3. Troubleshooting programme validation

The first slave was actively disconnected from the CAN bus, and alerts were observed from the red LEDs of both the master and the slave, as shown in Figure 18.

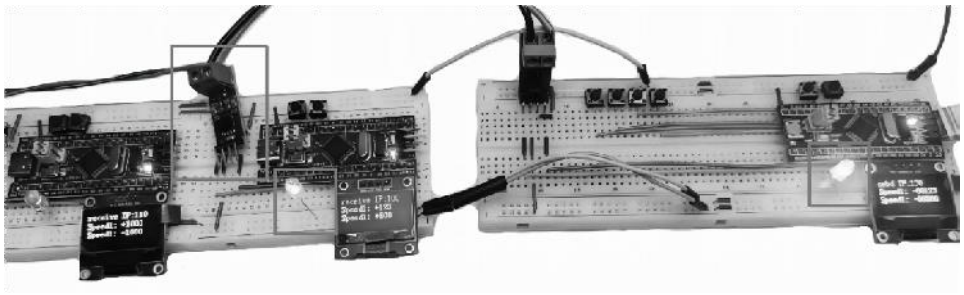

Figure 18. Disconnection of the first slave from the CAN bus network phenomenon

The host was actively disconnected from the CAN bus, triggering alarms from the red LEDs of both the host and all slaves. The host screen displayed the messages "Network disconnection" and "Error IP: 100," as shown in Figure 19.

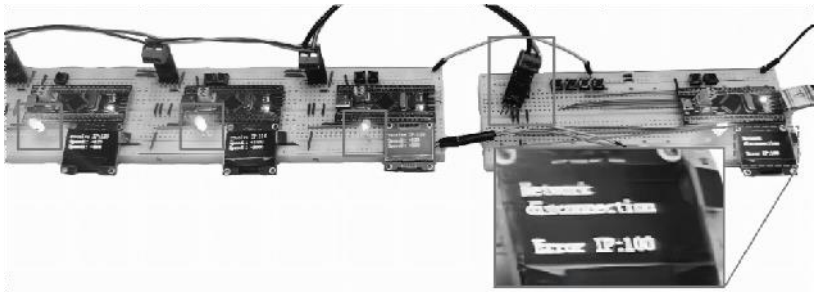

Figure 19. Bus Disconnect Phenomenon

Experimental verification shows that the serial multi-motor Communication programme based on the CAN bus master-slave architecture ensures reliable communication during operation and can promptly and effectively issue alarms according to the preset programme in case of failure.

6.Conclusion

This paper proposes a serial multi-motor Communication programme based on the CAN bus

master-slave architecture to address poor scalability and insufficient dynamic response in multi-motor cooperative control. Through modular hardware design, a dynamic node coding protocol, and a master-slave communication protocol, the system achieves flexible expansion and high-precision control of multi-motor systems. The master-slave architecture, supported by the dynamic node coding protocol, enables modular cascading of slaves, allowing the system to flexibly add or remove motor nodes as needed without manual IP address configuration, significantly improving deployment efficiency and scalability. Utilizing the CAN bus differential transmission and error self-test mechanism, combined with the host's periodic polling and PID closed-loop control algorithm, the system achieves precise control of multiple motors with minimal synchronization error, satisfying the demands of complex load scenarios. The system incorporates multiple fault-tolerant mechanisms, including automatic detachment of faulty nodes, dichotomous fault checking, and emergency broadcast instructions, enabling it to maintain core functions when individual nodes fail or the network is disrupted, thus ensuring system reliability in industrial applications. Additionally, the system can sustain core functions despite hardware damage. At the hardware level, the design is based on the STM32F103C8T6 and TJA1050 chips, delivering low power consumption and high real-time performance. At the software level, the slave driver communication protocol and dynamic IP allocation via Flash memory guarantee rapid system reconfiguration capabilities.

This programme has limitations in handling simultaneous access by multiple motors and depends on the response speed of individual slave fault checks. In the future, we aim to further optimize the conflict detection efficiency of the dynamic coding protocol and explore deterministic scheduling strategies based on time-triggered CAN (TTCAN) to meet more stringent industrial timing requirements. This research provides a cost-effective technical reference for distributed multi-motor control systems and holds broad engineering application prospects.

References

[1] R. Bader, A. Ali, and N. M. Mirza, "Ai and robotics leading industry 4.0," in 2022 9th International Conference on Internet of Things: Systems, Management and Security (IOTSMS), 2022: IEEE, pp. 1-4.

[2] Y. Zhang, B. Lu, and J. Chen, "The intelligent transformation path of Cultural Industrial Parks under science and technology empowerment," in 2021 International Conference on

Culture-oriented Science & Technology (ICCST), 2021: IEEE, pp. 124-128.

[3] P. S. Kholiya, A. Kapoor, M. Rana, and M. Bhushan, "Intelligent process automation: The future of digital transformation," in 2021 10th International Conference on System Modeling & Advancement in Research Trends (SMART), 2021: IEEE, pp. 185-190.

[4] P. Sadhu et al., "Smart Industrial Machine Management and Control System Based on IoT," in 2024 IEEE International Conference on Artificial Intelligence in Engineering and Technology (IICAIET), 2024: IEEE, pp. 471-476.

[5] S. Fu, H. Ren, T. Lin, S. Zhou, Q. Chen, and Z. Li, "SM-PI control strategy of electric motor-pump for pure electric construction machinery," IEEE Access, vol. 8, pp. 100241-100250, 2020.

[6] W. Ryu, "Implementation of dynamic node management in Hadoop cluster," in 2018 International Conference on Electronics, Information, and Communication (ICEIC), 2018: IEEE, pp. 1-2.

[7] A. Anakath, R. Kannadasan, G. S. Margarat, and A. P. Pandian, "Dynamic Topology Management in Ad-Hoc Networks for Improved Performance," in 2024 Second International Conference on Advances in Information Technology (ICAIT), 2024, vol. 1: IEEE, pp. 1-5.

[8] Peng nengling, Wei bo, Li zhenshan and Feng junqiang, "Research on electromagnetic compatibility performance of new energy bus CAN network," 2014 IEEE Conference and Expo Transportation Electrification Asia-Pacific (ITEC Asia-Pacific), Beijing, China, 2014, pp. 1-3, doi: 10.1109/ITEC-AP.2014.6940678.

[9] C. Zhang, "Research on the Real-time Performance of CAN Bus Based on 4WID New Energy Bus Network," 2021 International Conference on Information Control, Electrical Engineering and Rail Transit (ICEERT), Lanzhou, China, 2021, pp. 24-28, doi: 10.1109/ICEERT53919.2021.00013.

[10] A. M. Elshaer, M. M. Elrakaiby and M. E. Harb, "Autonomous Car Implementation Based on CAN Bus Protocol for IoT Applications," 2018 13th International Conference on Computer Engineering and Systems (ICCES), Cairo, Egypt, 2018, pp. 275-278, doi: 10.1109/ICCES.2018.8639206.

[11] C. Pengcheng and J. Zhicheng, "Simulation Study on Tracking Control of Mobile Robot Based on Cascaded Adaptive Approach*," 2007 Chinese Control Conference, Zhangjiajie, China, 2007, pp. 399-403, doi: 10.1109/CHICC.2006.4347319.

[12] Z. Chen, Y. -S. Hao, Z. -G. Su and L. Sun, "Cascade Disturbance Observer-Based Control Design for Cascaded Systems With Considerable Inner-Loop Dynamics," in IEEE Transactions on Automation Science and Engineering, vol. 22, pp. 12621-12632, 2025, doi: 10.1109/TASE.2025.3544563.

[13] M. T. S. Hugo, C. U. Oñate and J. K. Molina, "Driving a 3 DOF Robotic Manipulator with Protected Data through Industrial Communication using Master-Slave Architecture," 2019 IEEE 4th Colombian Conference on Automatic Control (CCAC), Medellin, Colombia, 2019, pp. 1-6, doi: 10.1109/CCAC.2019.8921340.

[14] M. E. Chamlee, E. W. Zegura and A. Mankin, "Design and evaluation of a protocol for automated hierarchical address assignment," Proceedings Ninth International Conference on Computer Communications and Networks (Cat.No.00EX440), Las Vegas, NV, USA, 2000, pp. 328-333, doi: 10.1109/ICCCN.2000.885510.