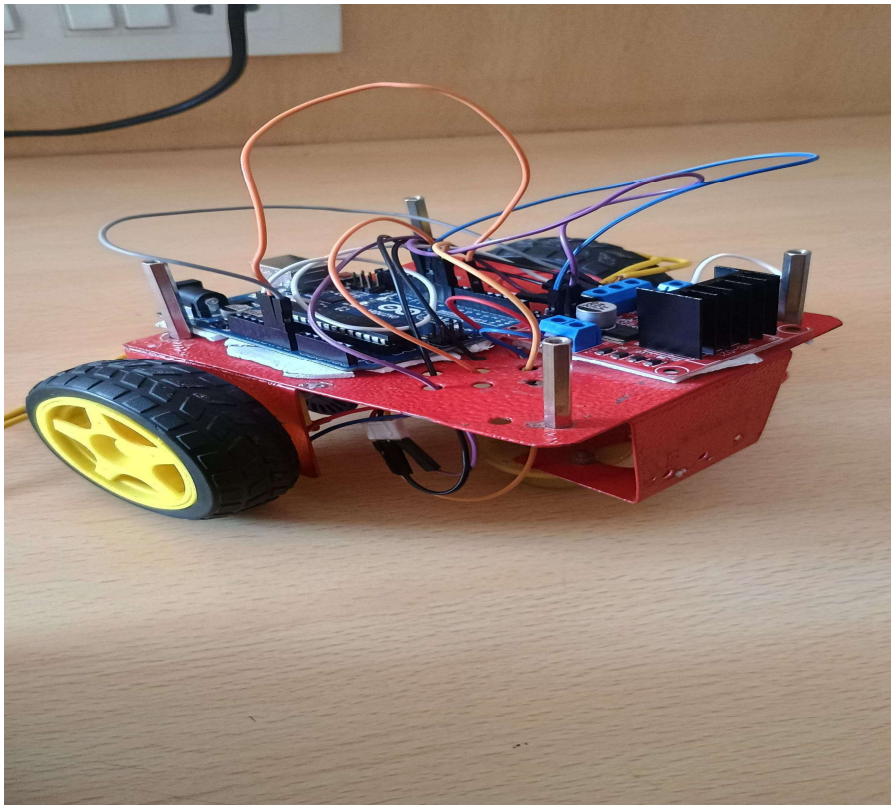
Automated RPM Balancer for Dual-Motor Systems Using FreeRTOS



LNMIIT
The LNM Institute of Information Technology

Kunal Sharma 21UEC080 ECE LNMIIT	Garvit Goyal 21DEC004 ECE LNMIIT

Submission Date: 7/12/2024

Automated RPM Balancer for Dual-Motor Systems Using FreeRTOS

Kunal Sharma 21UEC080 ECE LNMIIT	Garvit Goyal 21DEC004 ECE LNMIIT

Submission Date: 7/12/2024

Summary

The project titled "Automated RPM Balancer for Dual-Motor Systems Using FreeRTOS" focuses on enhancing the synchronization of dual motors in autonomous vehicles. The goal is to improve vehicle stability, especially in challenging conditions like uneven terrains or sharp turns. The system employs FreeRTOS for real-time task management, ensuring that critical tasks such as motor control and obstacle detection are handled promptly. Key components include optical encoders for speed feedback, DC motors for movement, and the L298N motor driver for controlling motor speed. The project also incorporates Pulse Width Modulation (PWM) to synchronize motor speeds, ensuring precise wheel motion.

The system is designed to dynamically adjust motor speeds based on encoder feedback, leveraging efficient memory management techniques and optimized task scheduling to ensure high performance. The project addresses synchronization challenges, providing a foundation for future advancements in autonomous vehicle technology.

Contents

Chapter 1

Introduction

1.1 Project Overview

The **Automated RPM Balancer for Dual-Motor Systems Using FreeRTOS** is designed to address synchronization challenges in dual-motor setups, particularly for autonomous vehicles where precise control over wheel motion is critical. Autonomous vehicles often rely on real-time responsiveness to maintain stability, especially in scenarios such as turning or navigating uneven terrain. The project utilizes FreeRTOS, a real-time operating system, to implement a robust and efficient control system for this purpose.

At the core of the system is the Adaptive Wheel Synchronization mechanism, which ensures that both wheels of the vehicle rotate at the same speed. This is achieved through the following technical implementations:

1. **Optical Encoder Integration**
   Optical encoders are employed to measure the rotational speed of each wheel. These encoders detect wheel motion by using an optical disc with a patterned design, placed between an infrared emitter and receiver. As the wheel rotates, the disc interrupts the IR beam, generating a pulse train corresponding to the rotation. These pulses are counted and converted into the RPM (Revolutions Per Minute) of each motor, providing precise feedback for speed control.

2. **Real-Time Task Management**
   Using FreeRTOS, the project benefits from real-time task scheduling and multi-tasking capabilities. Key features utilized include:

   - *Task Priority Management*: Ensures high-priority tasks, such as obstacle detection via IR sensors, are executed promptly.
   - *Task Yielding*: Tasks can yield the processor when idle, improving overall system responsiveness.
   - *Dynamic Task Suspension and Resumption*: When the IR sensor detects an obstacle, all motor-related tasks are suspended, prioritizing safety.

3. **PWM Control for Motors**
   Pulse Width Modulation (PWM) signals are used to control the speed of the DC motors. By dynamically adjusting the duty cycle of the PWM signals, the system synchronizes motor speeds, compensating for natural discrepancies between motors due to manufacturing differences or environmental conditions.

4. **Queue-Based Data Handling**
   The system utilizes FreeRTOS queues to store and transfer data between tasks. This approach prevents the corruption of static variables, enhances memory integrity, and improves system reliability.

5. **Efficient Memory Management**

   - The `uxTaskGetStackHighWaterMark()` API is used to monitor unused stack space for each task, enabling optimization of stack allocation. This reduces memory wastage and ensures that sufficient resources are available for critical tasks.
   - Memory usage statistics indicate that 34% of flash memory and 23% of SRAM are utilized, leaving ample space for future extensions.

6. **Adaptive Synchronization Algorithm**
   The core synchronization algorithm processes encoder feedback in real time to dynamically adjust the speed of each motor. By comparing the RPM of both motors, the system fine-tunes the PWM signals to equalize their speeds. This ensures precise wheel motion, contributing to vehicle stability.

This project demonstrates how FreeRTOS can be leveraged to implement an advanced embedded system with real-time control, optimized memory usage, and enhanced responsiveness. By addressing synchronization challenges in dual-motor systems, it sets the foundation for further innovations in autonomous vehicle technology.

1.2 Objective

The primary objective of this project is to develop a robust and adaptive control system that ensures synchronized motion of dual-motor systems in autonomous vehicles. Achieving precise synchronization between the two motors is critical for improving vehicle stability, maneuverability, and performance, particularly in challenging scenarios such as sharp turns, uneven terrains, or dynamic obstacle avoidance.

To accomplish this, the project focuses on the following goals:

1. **Enhance Vehicle Stability**
   By ensuring that both wheels rotate at the same speed, the system minimizes discrepancies that could destabilize the vehicle. This is particularly important during navigation over uneven surfaces or when executing sharp turns, where motor speed discrepancies can lead to reduced traction and control.

2. **Leverage FreeRTOS for Real-Time Responsiveness**
   FreeRTOS provides the capability to manage multiple tasks with varying priorities in a time-critical manner. This ensures prompt and efficient handling of tasks like motor speed control, obstacle detection, and synchronization feedback, all of which are essential for real-time performance.

3. **Adaptive Motor Synchronization**
   The system dynamically adjusts motor speeds by processing feedback from optical

encoders in real time. This adaptability ensures that synchronization is maintained under varying load conditions, motor wear, or environmental changes.

4. **Efficient Resource Management**
   Through optimized memory usage and stack management, the project aims to make the best use of limited microcontroller resources, leaving room for potential feature extensions or enhancements.

5. **Future-Ready Design**
   The system is designed with scalability and modularity in mind, allowing for integration with more complex control algorithms or additional sensors in the future. This ensures its applicability in advanced autonomous systems.

Overall, the project not only addresses the technical challenges associated with dual-motor synchronization but also sets a foundation for future advancements in autonomous vehicle technology. By combining precise motor control with real-time operating system capabilities, it delivers a practical and efficient solution to enhance the reliability and performance of embedded systems in real-world applications.

Chapter 2

Components Used

This section provides an overview of the components used in the project, their technical details, importance, and how they contribute to the overall functionality of the system.

2.1 Optical Encoder

An optical encoder is a sensor device used to measure the rotational speed of the motors by converting mechanical motion into electrical signals. It uses an optical disc with a patterned design placed between an infrared emitter and receiver. Interruptions in the IR beam caused by the rotating disc generate pulses corresponding to wheel rotation.

Importance in the Project:

- Measures the speed of each wheel with high accuracy.
- Provides feedback to adjust motor speeds dynamically, ensuring synchronization.

Key Specifications:

- Resolution: Number of pulses per revolution (PPR).
- Output: Digital signals (pulses).
- Response Time: Suitable for high-speed operations.

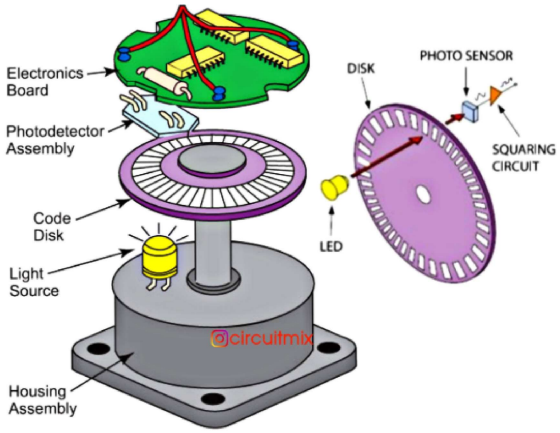


Figure 2.1: Optical Encoder

2.2 Arduino Uno

The Arduino Uno is a microcontroller board based on the ATmega328P. It serves as the primary processing unit for the project, managing tasks like reading encoder signals, generating PWM for motor control, and executing real-time operations via FreeRTOS.

Importance in the Project:

- Acts as the central control unit for executing tasks.
- Provides GPIO pins for interfacing with sensors and actuators.
- Supports FreeRTOS for multitasking and real-time performance.

Key Specifications:

- Processor: ATmega328P (16 MHz).
- Memory: 32 KB Flash, 2 KB SRAM, 1 KB EEPROM.
- Digital I/O Pins: 14 (6 with PWM output).

Figure 2.2: Arduino Uno

2.3 DC Motor

DC motors convert electrical energy into mechanical energy, enabling the rotation of the wheels. The speed of the motor is controlled using PWM signals generated by the microcontroller.

Importance in the Project:

- Provides the motion necessary for vehicle operation.
- Works in conjunction with encoders for speed control and synchronization.

Key Specifications:

- Operating Voltage: 6–12V.
- Speed: Up to 400 RPM (adjustable using PWM).

- Torque: Depends on motor type and load.

Figure 2.3: DC Motor

2.4 L298N Motor Driver

The L298N is a dual H-bridge motor driver module that allows independent control of two motors. It receives PWM signals from the microcontroller to regulate motor speed and direction.

Importance in the Project:

- Controls motor operation based on microcontroller inputs.
- Provides adequate current and voltage to drive motors efficiently.

Key Specifications:

- Operating Voltage: 5–35V.
- Output Current: Up to 2A per channel.
- Control: PWM and direction pins.

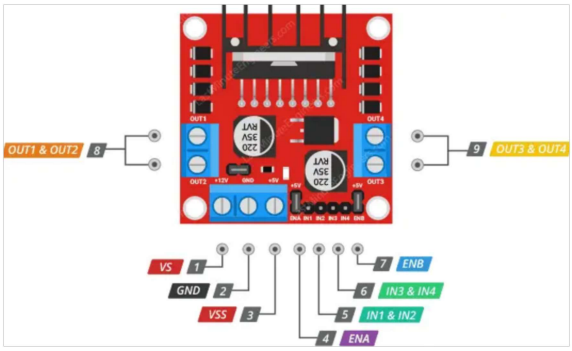


Figure 2.4: L298N Motor Driver

2.5 Steel Chassis

The steel chassis serves as the structural framework of the vehicle, housing all the components securely. It provides mechanical stability and durability, ensuring reliable operation under varying conditions.

Importance in the Project:

- Protects and supports the components.
- Ensures the system operates stably during movement.

Key Specifications:

- Material: High-strength steel.
- Design: Compact, with mounting slots for components.

Figure 2.5: Steel Chassis

2.6 Breadboard and Jumper Wires

The breadboard and jumper wires facilitate the interconnection of components for prototyping. They allow for flexible and non-permanent connections, enabling easy modifications during development.

Importance in the Project:

- Simplify wiring and component integration.
- Enable quick changes and testing during prototyping.

Key Features:

- Breadboard: Provides a grid of connection points.
- Jumper Wires: Available in male-to-male, male-to-female, and female-to-female configurations.

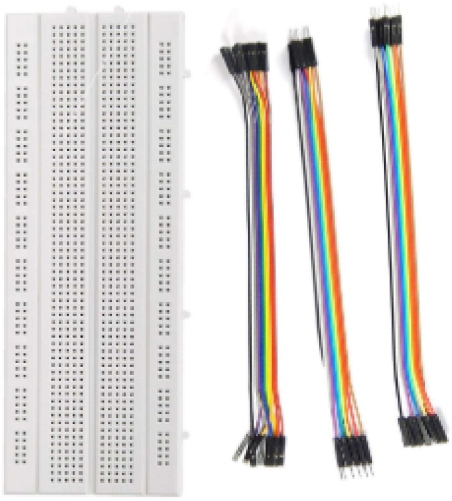


Figure 2.6: Breadboard and Jumper Wires

Chapter 3

Methodology

3.1 Overview

The methodology for the project focuses on achieving precise synchronization of dual motors in real time using FreeRTOS. The system is designed with modularity, adaptability, and efficiency in mind. It employs real-time task scheduling, feedback loops for motor control, and optimized memory management to ensure robust and reliable operation. The key steps in the methodology are detailed as follows.

3.2 System Design

The system is built around the integration of optical encoders, PWM motor control, and FreeRTOS for multitasking. Below are the main components of the design:

3.2.1 Feedback Mechanism

- Optical encoders measure the rotational speed (RPM) of each motor by generating pulse signals.
- The pulses are counted and converted into RPM values, which serve as feedback for the motor control algorithm.

3.2.2 Control Algorithm

- An adaptive synchronization algorithm compares the RPM of both motors in real time.
- The algorithm dynamically adjusts the duty cycle of PWM signals to equalize the speeds of the motors.

3.2.3 Task Management with FreeRTOS

- Tasks are created for individual operations, such as encoder data acquisition, motor control, and synchronization processing.
- Task priorities are assigned based on the criticality of operations.

- Queues are used to pass encoder data between tasks, ensuring thread-safe communication.

3.2.4 Memory Optimization

- The ‘uxTaskGetStackHighWaterMark()‘ API is used to monitor and optimize stack usage for each task.
- Efficient memory allocation is ensured by minimizing resource wastage and avoiding buffer overflows.

3.3 Implementation Steps

The system was implemented in the following steps:

3.3.1 Optical Encoder Integration

1. Optical encoders were mounted on each motor to measure their respective speeds.
2. Signals from the encoders were fed into interrupt pins of the microcontroller.
3. Pulse counts were converted to RPM values using a pre-calibrated formula.

3.3.2 PWM Motor Control

1. PWM signals were generated using the microcontroller's built-in timers.
2. The duty cycle of each PWM signal was adjusted dynamically based on encoder feedback.
3. The L298N motor driver module was used to drive the motors according to the PWM signals.

3.3.3 Task Scheduling in FreeRTOS

1. Tasks were created for key functions, including:
    - Reading encoder data.
    - Computing synchronization adjustments.
    - Generating PWM signals.
2. Queues were implemented for data communication between tasks to prevent variable corruption.
3. Task priorities ensured real-time responsiveness for critical operations.

3.3.4 System Calibration

1. The system was calibrated to account for discrepancies between motors caused by manufacturing tolerances.
2. Baseline RPM values were established under no-load conditions for both motors.
3. Fine adjustments to the synchronization algorithm were made to achieve optimal performance under varying loads.

3.4 Block Diagram

The block diagram in Figure 3.1 illustrates the flow of data and control signals within the system.

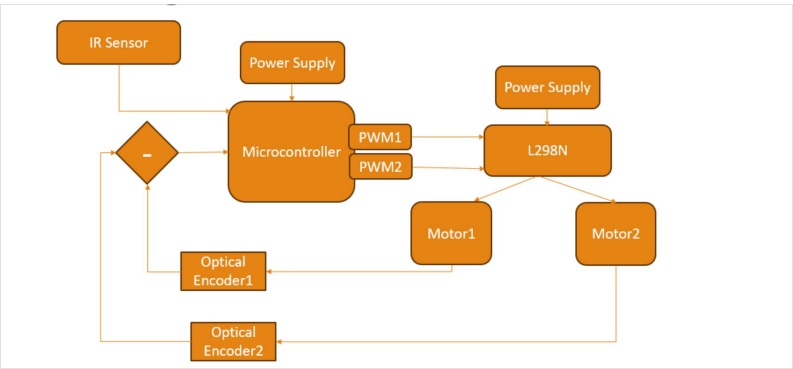


Figure 3.1: System Block Diagram

3.5 Testing and Validation

3.5.1 Simulation

- The control algorithm was tested using simulated encoder data to validate its performance.
- Boundary cases, such as extreme speed discrepancies, were tested to ensure robustness.

3.5.2 Hardware Testing

- The system was implemented on a physical dual-motor setup.
- RPM synchronization was verified using a tachometer for independent validation.
- The system's response to varying loads and obstacles was observed and fine-tuned.

3.5.3 Performance Metrics

Key metrics, including synchronization accuracy, power consumption, and task execution times, were recorded and analyzed. These metrics demonstrated the effectiveness of the methodology in achieving precise motor synchronization and efficient resource utilization.

Chapter 4

Observations and Results

4.1 Observations

- Optical encoders detect wheel motion via an optical disc, which generates pulses corresponding to rotational movement.
- Accurate tracking of wheel speed enables precise control.

4.2 Performance Metrics

Power Consumption:

- Idle State: 0.27W
- Running State: 10.53W

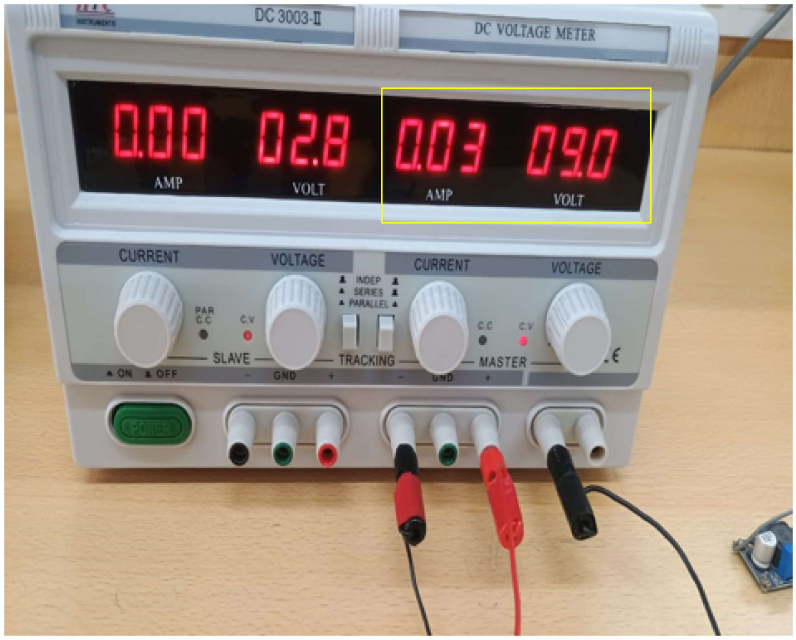


Figure 4.1: Idle State Power Consumption

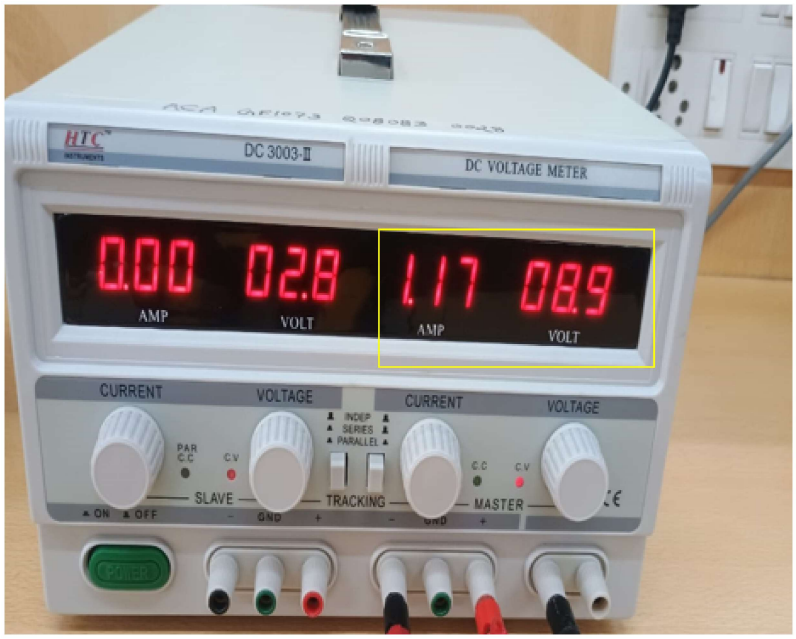


Figure 4.2: Running State Power Consumption

Memory Usage:

- Flash Memory: 34% utilized (11,222 bytes used).
- SRAM: 23% utilized (475 bytes used).

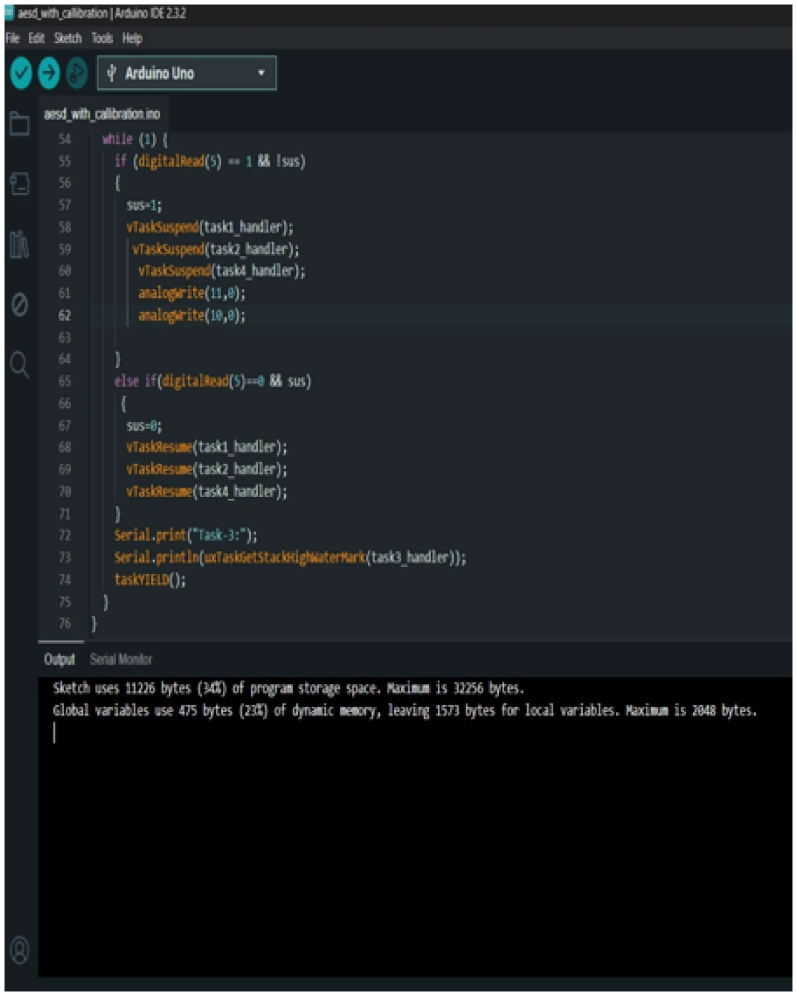


Figure 4.3: Sketch Memory Consumption

4.3 Optimized Stack Size

The FreeRTOS API ‘uxTaskGetStackHighWaterMark()‘ helped optimize memory usage. Benefits included reduced memory footprint and improved system stability.

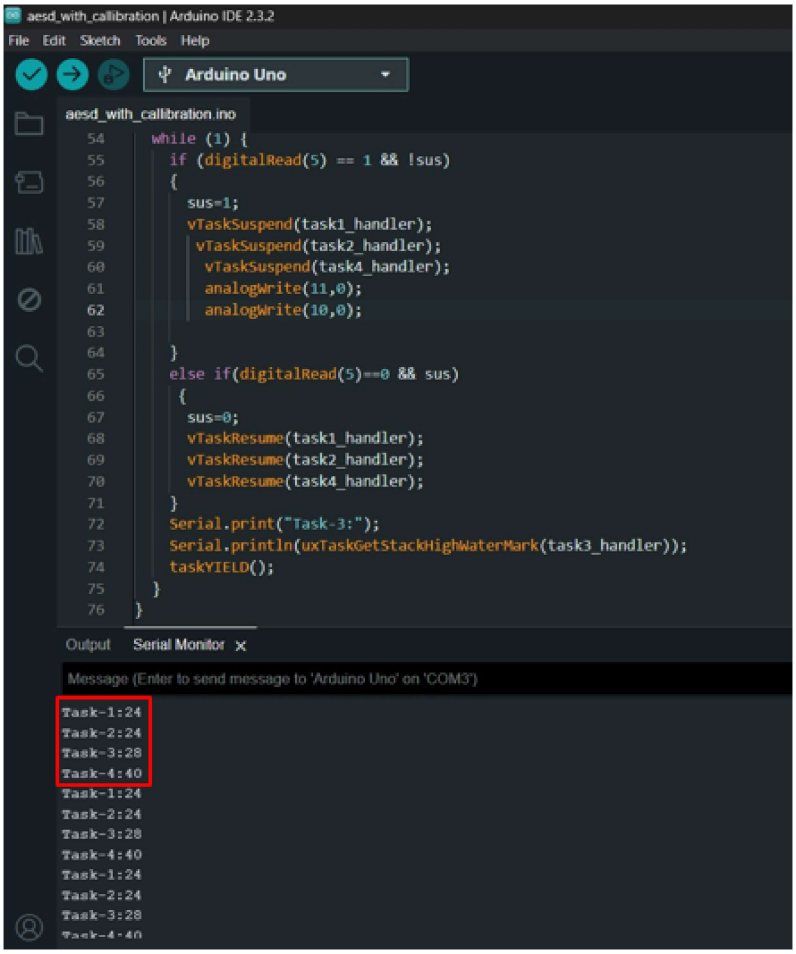


Figure 4.4: Stack Memory Optimization

Chapter 5

Problems Faced

5.1 Challenges Encountered During Development

During the development of the **Automated RPM Balancer for Dual-Motor Systems Using FreeRTOS**, several technical and practical challenges were encountered. These issues not only tested the robustness of the design but also necessitated iterative debugging and optimization. The following are the detailed challenges faced and the solutions implemented to address them:

5.1.1 Synchronization Issues Due to Non-Identical Motors

Problem: The motors used in the system, despite being of the same model, exhibited natural discrepancies in speed due to manufacturing tolerances and wear. These differences resulted in asynchronous wheel motion, which affected the vehicle's stability and maneuverability.

Impact: - Uneven motor speeds caused the vehicle to veer off its intended path. - Increased difficulty in maintaining precise control during sharp turns or when navigating uneven terrain.

Solution: - Implemented real-time feedback from optical encoders to dynamically adjust the PWM duty cycle for each motor. - Developed an adaptive synchronization algorithm to fine-tune motor speeds and maintain uniform rotation.

5.1.2 Memory Optimization and Variable Corruption

Problem: The use of global and static variables led to unexpected behavior in task execution. Some variables were overwritten or corrupted due to simultaneous access by multiple tasks, particularly in the absence of proper synchronization mechanisms.

Impact: - Erratic motor control and synchronization failures. - System instability, requiring frequent resets and debugging.

Solution: - Replaced global variables with FreeRTOS queues to safely transfer data between tasks. - Used mutexes to synchronize access to shared resources, ensuring data integrity. - Monitored memory usage using the `uxTaskGetStackHighWaterMark()` API to optimize stack allocation and prevent overflows.

5.1.3 Impact of Optical Encoder Switching Speed

Problem: The optical encoders used to measure wheel speed generated high-frequency pulse trains at higher motor speeds. This overwhelmed the microcontroller's interrupt handling capability, leading to missed or delayed pulses.

Impact: - Inaccurate RPM readings, affecting synchronization. - Reduced system responsiveness during rapid speed changes.

Solution: - Configured hardware interrupts to prioritize encoder signals over less critical tasks. - Introduced a debouncing mechanism in software to filter out spurious signals and reduce the load on the microcontroller.

5.1.4 PWM Resolution Limitations

Problem: The microcontroller's PWM resolution was insufficient for achieving precise speed adjustments at low RPMs, where minor changes in the duty cycle resulted in significant speed differences.

Impact: - Difficulty in achieving fine-grained motor control. - Reduced system efficiency and increased power consumption.

Solution: - Experimented with different PWM frequencies to balance resolution and system performance. - Introduced a proportional-integral (PI) control loop to achieve smoother speed adjustments.

5.1.5 Temperature Variations Affecting Motor Performance

Problem: Prolonged operation of the motors caused temperature-induced performance degradation, altering their speed and torque characteristics.

Impact: - Increased speed mismatches between motors. - Reduced synchronization accuracy over extended operation.

Solution: - Implemented a runtime calibration mechanism to periodically recalibrate motor parameters based on encoder feedback. - Enhanced the system's thermal design to improve heat dissipation.

5.1.6 Hardware Integration and Signal Noise

Problem: Interference from the motor's electromagnetic emissions caused noise in the encoder and PWM signals, leading to unreliable system behavior.

Impact: - Fluctuating speed measurements and control signals. - Difficulty in debugging and isolating the root cause of issues.

Solution: - Added hardware filters (capacitors and resistors) to suppress noise in encoder signals. - Used shielded cables and proper grounding techniques to minimize EMI effects.

5.1.7 Task Priority Conflicts in FreeRTOS

Problem: Improper task priority assignments caused high-priority tasks, such as obstacle detection, to preempt critical motor control tasks. This resulted in delayed synchronization updates.

Impact: - Reduced responsiveness to motor speed changes. - Inconsistent performance during high-priority task execution.

Solution: - Re-evaluated and optimized task priority levels to ensure smooth execution of critical tasks. - Implemented task-yielding and suspension mechanisms to improve overall system responsiveness.

5.2 Lessons Learned

The challenges faced during this project provided valuable insights into embedded system design and real-time operating system (RTOS) usage. Key takeaways include: - The importance of robust synchronization mechanisms, such as queues and mutexes, for ensuring data integrity. - The need for hardware and software solutions to mitigate the impact of noise and environmental variations. - The critical role of task management in achieving real-time responsiveness and system reliability.

By addressing these issues, the project team was able to deliver a stable and efficient dual-motor control system, setting the stage for future enhancements and scalability.

Chapter 6

Conclusion

6.1 Project Overview

The development of the **Automated RPM Balancer for Dual-Motor Systems Using FreeRTOS** has been a comprehensive exploration of embedded system design, real-time operating system (RTOS) implementation, and motor control optimization. The project successfully achieved its primary objective of synchronizing two motors to maintain uniform rotational speeds under varying conditions.

6.2 Key Achievements

The following milestones highlight the major accomplishments of the project:

6.2.1 Precision Motor Synchronization

- Developed and implemented a feedback-based control mechanism using optical encoders to ensure real-time synchronization of motor speeds. - Achieved stable operation even with inherent discrepancies between motors due to manufacturing tolerances or operational wear.

6.2.2 Robust RTOS Implementation

- Leveraged FreeRTOS to create a multitasking environment, ensuring efficient management of motor control, data acquisition, and synchronization tasks. - Effectively utilized RTOS features such as queues, mutexes, and task prioritization to optimize performance and ensure system stability.

6.2.3 Improved System Reliability

- Mitigated challenges arising from signal noise, memory constraints, and task conflicts through iterative debugging and optimization. - Ensured long-term reliability by addressing thermal effects on motor performance and incorporating runtime recalibration mechanisms.

6.2.4 Scalability and Modularity

- Designed the system with scalability in mind, enabling easy adaptation to more complex scenarios, such as multi-motor synchronization or integration with additional sensors and actuators. - Emphasized modular design principles to facilitate future upgrades and maintenance.

6.3 Challenges and Lessons Learned

The project provided valuable insights into the complexities of embedded system design and motor control. Overcoming challenges such as synchronization discrepancies, signal noise, and task conflicts enriched the team's understanding of real-world constraints and solutions. Key lessons learned include:

- The critical importance of feedback mechanisms in maintaining system accuracy and reliability. - The need for robust hardware design to minimize environmental interference and ensure consistent signal quality. - The value of proper task management and resource allocation in RTOS-based systems.

6.4 Future Prospects and Applications

The success of this project opens the door to various potential applications and future enhancements:

6.4.1 Applications

- **Autonomous Robotics:** The synchronization mechanism can be extended to robotic platforms requiring precise control of multiple motors, such as robotic arms or mobile robots. - **Electric Vehicles:** The system's ability to balance dual-motor RPMs makes it applicable in electric vehicles with dual-motor drive systems, ensuring efficient power distribution. - **Industrial Automation:** The principles developed in this project can be utilized in conveyor belt systems, assembly lines, and other automation setups requiring synchronized motor operation.

6.4.2 Future Enhancements

- Integration of more advanced sensors, such as gyroscopes and accelerometers, for enhanced feedback and stability control. - Implementation of machine learning algorithms to predict and compensate for motor performance variations dynamically. - Expansion of the system to support multi-motor synchronization and more complex motion patterns.

6.5 Conclusion

This project has demonstrated the feasibility and effectiveness of using FreeRTOS for precise dual-motor synchronization. By addressing both technical and practical challenges, the team successfully delivered a reliable and scalable solution. The knowledge and skills gained from this endeavor provide a solid foundation for future advancements in embedded system design and real-time control applications. The project underscores

the potential of innovative solutions to meet the demands of modern automation and robotics industries.

Chapter 7

References

1. https://www.freertos.org/
2. https://cdn-shop.adafruit.com/product-files/3986/ee-sx47_67_ds_e_13_2_csm483
3. Advanced Embedded Systems Design Course by Dr. Deepak Nair