Fachhochschule
Dortmund
University of Applied Sciences and Arts

Fachbereich Informatik

Analyse der Echtzeitfähigkeit von Micro-ROS und FreeRTOS am Beispiel einer Robotersteuerungssoftware

Bachelorarbeit

Zijian Xu
Matrikelnummer: 7204211

Version vom: 14. März 2025
Betreuer: Prof. Dr. Christof Röhrig
M. Sc. Alexander Miller

Kurzfassung

Diese Arbeit analysiert die Echtzeitfähigkeit von Micro-ROS und FreeRTOS am Beispiel einer Robotersteuerungssoftware. Ziel ist es, die Performance beider Systeme im Hinblick auf Ausführungszeiten, Ressourcenverbrauch und Echtzeitverhalten zu vergleichen.

Die Analyse beginnt zuerst mit der vollständigen Umstellung der bestehenden Robotersteuerungssoftware von Micro-ROS auf FreeRTOS. Anschließend wird die Data Watchpoint and Trace Unit (DWT) Zur Analyse eingesetzt, um eine zyklengetreue Erfassung des Programmlaufs zu ermöglichen.

Abschließend wird das Ergebnis evaluiert, welches unter anderem die Ausführungszeiten von FreeRTOS-Prozessen, zeitkritischen Funktionen sowie das Verhältnis von Ausführung zu Leerlaufzeit umfasst. Die Ergebnisse sollen Einsichten darüber geben, inwieweit Micro-ROS und FreeRTOS für Echtzeitanwendungen in der Robotik geeignet sind und welche Vor- oder Nachteile die jeweiligen Systeme bieten.

Abstract

Inhaltsverzeichnis

Abbildungsverzeichnis

Quellcodeverzeichnis

Abkürzungsverzeichnis

DWT Data Watchpoint and Trace Unit

ROS Robot Operating System

DDS Data Distribution Service

Einleitung

Die vorliegende Arbeit beschäftigt sich mit der Analyse der Echtzeitfähigkeit von Micro-ROS und FreeRTOS am Beispiel einer Robotersteuerungssoftware. Ziel ist es, die Performance beider Systeme hinsichtlich der Ausführungszeiten, Ressourcenverbrauch sowie Echtzeitverhalten zu untersuchen, um ihre Eignung für Roboteranwendungen zu bewerten.

Die Arbeit beinhaltet schwerpunktmäßig die Entwicklung einer Methode zum Profiling der Steuerungssoftware eines mobilen Roboters. Dabei wird zunächst die bestehende Firmware, die auf Micro-ROS basiert, im Rahmen dieser Arbeit auf FreeRTOS portiert. Anschließend wird die Methodik zur Generierung von Profiling-Daten für die Analyse festgelegt und implementiert, und die resultierende Ergebnisse evaluiert.

Zu Beginn wird ein Überblick über die grundlegenden Konzepte gegeben. Darauffolgend werden die Implementierungen detailliert beschrieben. Abschließend werden die erzielten Ergebnisse vorgestellt und bewertet, und es wird ein Ausblick auf weitere Anwendungsmöglichkeiten und Optimierungspotenziale gegeben.

1 Hintergrund

Die vorliegende Bachelorarbeit hat zum Ziel, die Robotersteuerungssoftware, die derzeit auf Micro-ROS basiert, auf FreeRTOS zu portieren, um einen vergleichenden Leistungsanalyse zwischen beiden Plattformen durchzuführen. Beide Systeme sind für die Steuerung eines mobilen Roboters auf einem Cortex-M7 Mikrocontroller von Arm konzipiert, unterscheiden sich jedoch in ihrer grundlegenden Architektur, was sich auch in ihrer Echtzeitfähigkeit und Ressourcennutzung widerspiegelt. Während Micro-ROS auf der Robot Operating System (ROS)2 aufbaut und eine höhere Abstraktionsebene sowie standardisierte Kommunikationsschnittstellen mittels der Data Distribution Service (DDS)-Middleware bietet, basiert Micro-ROS selbst auf FreeRTOS. Die Portierung der Robotersteuerungssoftware von Micro-ROS auf FreeRTOS kann daher als eine Reduzierung der Abhängigkeitsebene betrachtet werden. Dies ermöglicht eine direktere und effizientere Nutzung der zugrunde liegenden Echtzeit-, sowie Speicherressourcen.

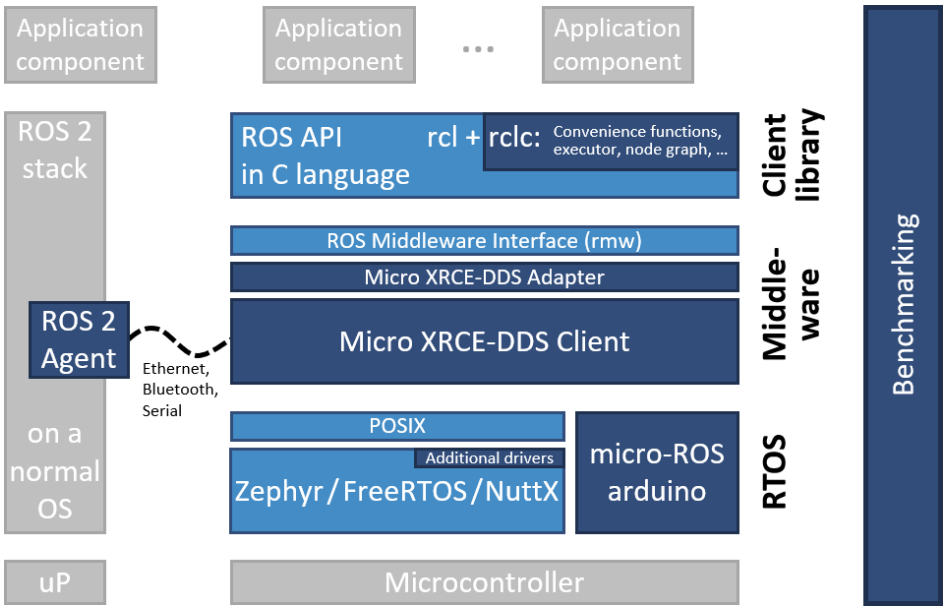

Abbildung 1: Micro-ROS Architektur[Kou23, S. 6]

Nach dem Wechsel zu FreeRTOS wird die Echtzeitleistung der Steuerungssoftware analysiert mit einem besonderen Fokus auf den Overhead, der durch die Micro-ROS-Schicht verursacht wird. Der Vergleich soll aufzeigen, inwiefern FreeRTOS durch die Eliminierung dieser zusätzlichen Abhängigkeit eine effizientere und leichtgewichtige Lösung für kritische Roboteranwendungen darstellt. Dabei soll der Einsatz einer zyklengenaue Messung des Programmablaufs ermöglichen, fundierte Aussagen über die Echtzeitfähigkeit beider Plattformen zu treffen, und den Leistungsgewinn anhand von diesem Beispiel für eine Steuerungssoftware quantitativ zu belegen.

1.1 FreeRTOS

FreeRTOS ist ein leichtgewichtiges, Open-Source-Echtzeitbetriebssystem, das speziell für eingebettete Systeme entwickelt wurde. Es zeichnet sich unter anderem durch deterministisches Verhalten mit Echtzeitgarantie sowie konfigurierbare Allokation Speicher aus. Diese Eigenschaften machen es zu einer idealen Wahl für Robotersteuerungssoftware, insbesondere wenn Echtzeitanforderungen und effiziente Ressourcennutzung im Vordergrund stehen.

Im Rahmen dieser Bachelorarbeit wird FreeRTOS als Alternative zu Micro-ROS eingesetzt, um die bestehende Robotersteuerungssoftware zu portieren. Während Micro-ROS auf der Robot Operating System (ROS) 2-Middleware basiert und eine hohe Abstraktionsebene sowie standardisierte Kommunikationsschnittstellen bietet, ist FreeRTOS darauf ausgelegt, deterministische Echtzeitanforderungen zu erfüllen und kritische Aufgaben in eingebetteten Systemen effizient zu verwalten.

Ziel der Migration ist es, die Echtzeitleistung beider Systeme zu vergleichen, insbesondere in Bezug auf Latenzzeiten, Scheduling-Effizienz und Speicherauslastung. FreeRTOS bietet hierfür geeignete Mechanismen wie Trace Hooks, die eine zyklengenaue Messung der Systemleistung ermöglichen. Diese Analyse soll aufzeigen, inwiefern FreeRTOS als Alternative zu Micro-ROS für kritische Roboteranwendungen geeignet ist und welche Vorteile es in Bezug auf Determinismus und Ressourceneffizienz bietet.

1.1.1 Konzepte und Komponenten

1.1.2 Konzepte

Hier ist die Sektion Konzepte und Komponenten unter Berücksichtigung der gegebenen Anforderungen:

Konzepte und Komponenten FreeRTOS bietet eine Reihe von grundlegenden Konzepten und Komponenten, die für die Entwicklung von eingebetteten Echtzeitsystemen essenziell sind. Im Fokus dieser Arbeit stehen insbesondere Queues, Semaphore/Mutexe, Direct Task Notifications und Trace Hooks, die für die effiziente Steuerung und Synchronisation von Robotersteuerungssoftware von zentraler Bedeutung sind.

Queues Queues sind eine der Kernkomponenten von FreeRTOS und dienen der Interprozesskommunikation zwischen Tasks. Sie ermöglichen den sicheren Austausch von Daten und Nachrichten, selbst in multithreaded Umgebungen. Queues sind thread-

sicher und können sowohl zur Datenübertragung als auch zur Task-Synchronisation verwendet werden. Viele der höheren Synchronisationsmechanismen in FreeRTOS, wie Semaphore und Mutexe, sind auf Queues aufgebaut, was ihre zentrale Rolle im System unterstreicht.

Semaphore und Mutexe Semaphore und Mutexe sind Synchronisationsmechanismen, die den Zugriff auf gemeinsame Ressourcen koordinieren und Race Conditions verhindern.

Semaphore werden verwendet, um den Zugriff auf Ressourcen zu beschränken oder um Ereignisse zwischen Tasks zu signalisieren. Mutexe (Mutual Exclusion) sind spezialisierte Semaphore, die sicherstellen, dass nur ein Task gleichzeitig auf eine kritische Ressource zugreifen kann.

Direct Task Notifications Direct Task Notifications sind ein effizienter und ressourcensparender Mechanismus zur Task-Synchronisation. Im Gegensatz zu Semaphore oder Queues senden sie direkte Signale an einen Task, ohne zusätzliche Datenstrukturen zu benötigen. Dies reduziert den Overhead und ermöglicht eine schnellere Synchronisation.

Vergleich mit Semaphore: Während Semaphore universell einsetzbar sind, bieten Direct Task Notifications eine leistungsstarke Alternative, wenn nur einfache Synchronisation zwischen Tasks erforderlich ist.

Trace Hooks Trace Hooks sind spezielle Vorhakpunkte in der FreeRTOS-API, die es ermöglichen, Ereignisse im System zu verfolgen und zu protokollieren. Sie sind besonders nützlich für die Echtzeitanalyse, da sie detaillierte Informationen über Task-Wechsel, Scheduling und Synchronisationsereignisse liefern. Diese Funktionen sind entscheidend, um die Leistungscharakteristika der Robotersteuerungssoftware zu analysieren und Optimierungspotenziale zu identifizieren.

Zusammenfassend bilden diese Komponenten die Grundlage für die effiziente und deterministische Steuerung von Robotersteuerungssoftware in FreeRTOS. Sie ermöglichen eine zuverlässige Kommunikation, Synchronisation und Analyse, die für die erfolgreiche Migration und den Vergleich mit Micro-ROS unerlässlich sind.

1.2 Echtzeitanalyse

Um die Echtzeitanalyse der Steuerungssoftware durchzuführen, ist eine Methode erforderlich, die jeden Ausführungsabschnitt des Codes flexibel, präzise und threadsicher messen kann. Da die Software multithreaded ist, muss sichergestellt werden, dass die Messungen trotz Interrupts und preemptivem Scheduling korrekt und zyklengetreu durchgeführt werden können.

1.3 Zyklenzähler

Basierend auf den oben genannten Herausforderungen bietet die Data Watchpoint and Trace Unit (DWT) eine geeignete Lösung [ARMe]. Die DWT ist ein Hardwaremodul in Prozessoren inklusive ARMv7-M, das speziell für die Analyse von Programmabläufen entwickelt wurde [ARMa]. Ein für diese Arbeit zentraler Bestandteil der DWT ist der Zyklenzähler `DWT_CYCCNT`, der bei jedem Takt inkrementiert wird, solange sich der Prozessor nicht im Debug-Zustand befindet [ARMb]. Dadurch ermöglicht die DWT beispielsweise die Erfassung von Echtzeitaspekten mit zyklengenauer Präzision under normaler Operation [ARMc].

1.3.1 Beispiel: SEGGER SystemView

Ein Beispiel hierfür ist SEGGER SystemView, ein Echtzeit-Analysewerkzeug, das den DWT-Zyklenzähler einsetzt, um Live-Code-Profiling auf eingebetteten Systemen durchzuführen [SEG].

Das SEGGER SystemView nutzt den DWT, indem die Funktion `SEGGER_SYSVIEW_GET_TIMESTAMP()` einfach die hardkodierte Registeradresse des Zyklenzählers zurückgibt, anstatt die interne Funktion `SEGGER_SYSVIEW_X_GetTimestamp()` aufzurufen [SiFa, Armd]. Dies erfolgt, sofern das Macro `SEGGER_SYSVIEW_CORE_CM3` definiert ist [SiFb]. Aus dieser Definition wird gezeigt, dass die Nutzung des DWT für Cortex-M3, M4 sowie M7 Prozessoren aktiviert wird. Daraus könnte sich eventuell auch schlussfolgern lassen, dass der DWT nur auf diesen Prozessoren verfügbar ist.

Aufgrund dieser Eigenschaften der DWT wird die Implementierung für die Echtzeitanalyse auf dieser Komponente aufbauen.

2 Vorbereitung

2.1 Umstellung auf FreeRTOS

lorem

3 Abschluss

3.1 Fazit

3.2 Ausblick

Literaturverzeichnis

[ARMa] ARM Holdings: *ARMv7-M Architecture Reference Manual*. https://developer.arm.com/documentation/ddi0403/d/Debug-Architecture/ARMv7-M-Debug/The-Data-Watchpoint-and-Trace-unit/Profiling-counter-support?lang=en. – Zugriff: 14. März 2025

[ARMb] ARM Holdings: *ARMv7-M Architecture Reference Manual*. https://developer.arm.com/documentation/ddi0403/d/Debug-Architecture/ARMv7-M-Debug/The-Data-Watchpoint-and-Trace-unit/CYCCNT-cycle-counter-and-related-timers?lang=en. – Zugriff: 14. März 2025

[ARMc] ARM Holdings: *ARMv7-M Architecture Reference Manual*. https://developer.arm.com/documentation/ddi0403/d/Debug-Architecture/ARMv7-M-Debug/The-Data-Watchpoint-and-Trace-unit?lang=en. – Zugriff: 14. März 2025

[Armd] Arm Holdings: *Data Watchpoint and Trace Unit (DWT) Programmer's Model*. https://developer.arm.com/documentation/ddi0439/b/Data-Watchpoint-and-Trace-Unit/DWT-Programmers-Model. – Zugriff: 14. März 2025

[ARMe] ARM Holdings: *Summary: How many instructions have been executed on a Cortex-M processor?* https://developer.arm.com/documentation/ka001499/latest/. – Zugriff: 14. März 2025

[Kou23] Koubaa, Anis: *Robot Operating System (ROS) The Complete Reference*. Volume 7. Springer Verlag, 2023. – ISBN 978–3–031–09061–5

[SEG] SEGGER Microcontroller: *What is SystemView?* https://www.segger.com/products/development-tools/systemview/technology/what-is-systemview#how-does-it-work. – Zugriff: 14. März 2025

[SiFa] SiFive: *Segger_SystemView-metal Repository: SEGGER_SYSVIEW_Conf.h*. https://github.com/sifive/Segger_SystemView-metal/blob/cd89dbc2f3d5222d9856780517261ad7a2804805/SystemView/Config/SEGGER_SYSVIEW_Conf.h#L138. – Zugriff: 14. März 2025

[SiFb] SiFive: *Segger_SystemView-metal Repository: SEGGER_SYSVIEW_Conf.h*. https://github.com/sifive/Segger_SystemView-metal/blob/cd89dbc2f3d5222d9856780517261ad7a2804805/SystemView/Config/SEGGER_SYSVIEW_Conf.h#L79. – Zugriff: 14. März 2025