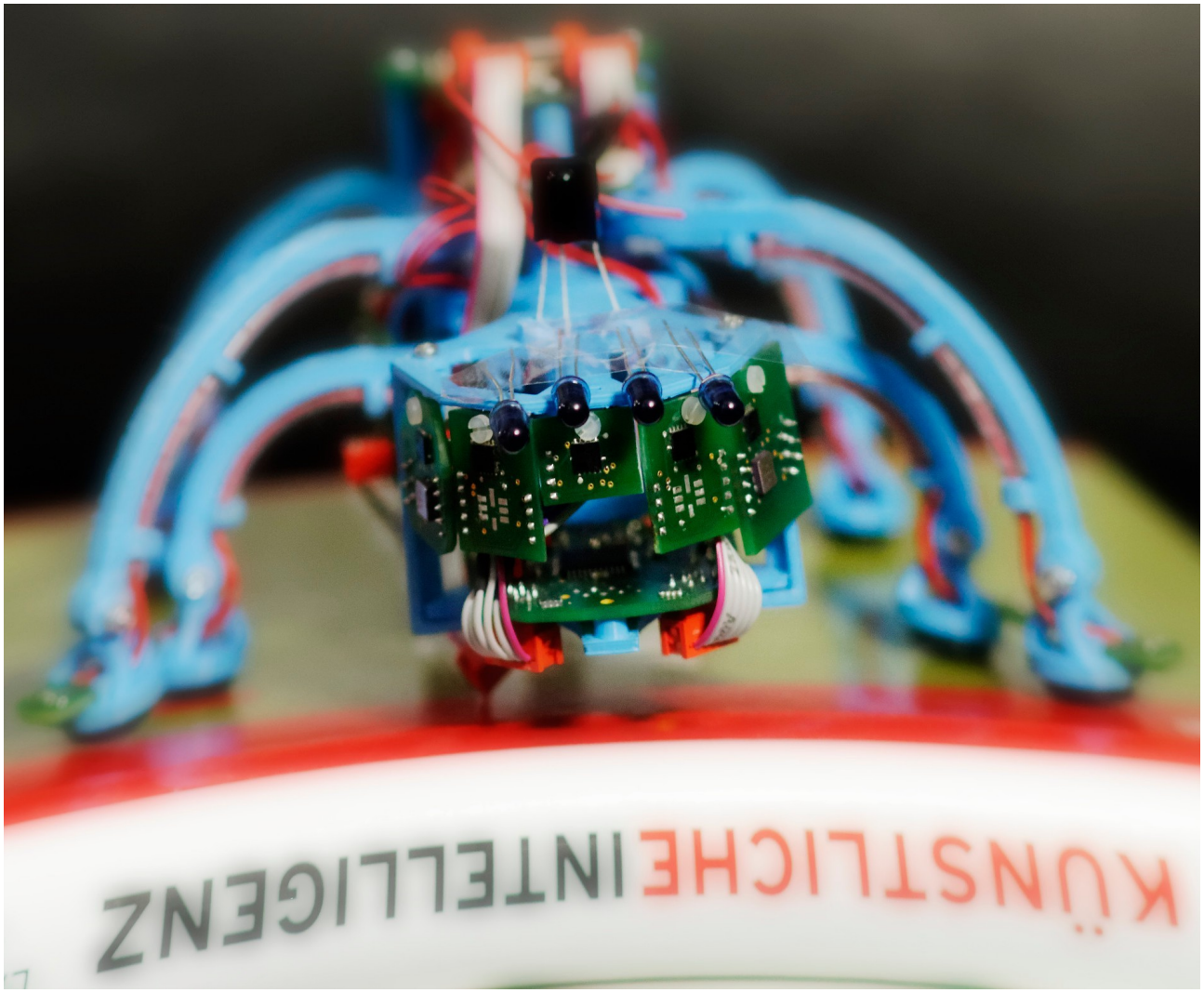
Konzept zu Projektarbeit „Interne Roboterkommunikation“
Sommersemester 2023

Table of Contents



1 Aufgabenstellung

Ziel der Projektarbeit ist die Portierung von Internen Roboterkommunikation in RoboControl Firmware von C auf (Micro) Python. Als Zielplattform ist der Raspberry Pi Pico vorgesehen. Dazu ist zuerst eine geeignete PIO basierte USART Implementierung zu erstellen. Im zweiten Schritt ist das vorhandene Kommunikationsmodul im RoboControl Firmware auf (Micro-) Python zu portieren.

1.1 Was ist RoboControl ?

RoboControl ist ein modulares Betriebssystem für kleine Roboter. Hierbei wird der Roboter selbst in kleinere Einheiten sog. ***Devices*** aufgeteilt, wobei jedes dieser Devices eine festgelegte Teilfunktion innerhalb des Roboters implementiert. Dies können z.b. Beinsteuerung, Kopfsensoren, Beinsensoren usw. sein. In der Regel wird in der Hardware jedes dieser Devices durch einen Mikrocontroller repräsentiert. Diese Devices können sowohl untereinander, als auch mit der Hauptlogik des Roboters über ein Baum/Sternförmiges Kommunikationsnetzwerk Daten untereinander austauschen. Zu diesem Zweck wurde in der Vergangenheit ein auf diese Kommunikationsart ausgerichtetes Protokoll entwickelt.

1.2 Kommunikationsmedium

Die Kommunikation zwischen den einzelnen Mikrocontrollern des Roboters erfolgt mittels seriellen Schnittstelle über einem USART. Dabei hat sich die 9Bittige Übertragungsart als einfachste und effizienteste erwiesen. Die im unterschied zu einem UART vorhandene zusätzliche Taktleitung ermöglicht es deutlich höhere Datenraten zu verwenden. Zusätzlich dazu werden Übertagungsfehler aufgrund Differenz in den tatsächlichen Taktraten der verwendeten Mikrocontroller quasi ausgeschlossen. Das 9te Bit wird verwendet um den Anfang und das Ende einer Nachricht zu kennzeichnen. Für unterschiedliche Pakettypen (Command, Message, Stream usw.) werden verschiedene Anfang bzw. Synchronisationszeichen verwendet.

1.3 Programmable Input Output

Bei der PIO handelt es sich um einem sehr einfachen Coprozessor mit vier unabhängigen State maschinen. Dieser ist hauptsächlich dazu gedacht um fehlende Schnittstellen in dem Pi Pico nachrüsten. Die Programmierung des PIO erfolgt in seinen speziellen Assembler befehlen (9 verschiedene Befehle). Diese Befehle können direkt aus einem Python Programm einzeln in dem PIO geschrieben werden wonach die State Maschie des PIO selbständig dass auf diese weise

entstandenes Programm ausführt. Das Programmspeicher aller State Maschined sind ist auf 32 Befehle beschränkt. Folgende Befehle stehen zu Verfügung:

IN, OUT , PUSH, PULL, MOV, IRQ, SET, WAIT, JMP

Das herausschieben der Daten erfolgt über den OSR (Output Shift Register) und lesen über den ISR (Input Shift Register). Die Kommunikation mit dem Core des Prozessors wird über eine TX- sowie RX-FiFo geregelt.

https://www.heise.de/blog/I-O-on-Steroids-PIO-die-programmierbare-Ein-Ausgabe-des-Raspberry-Pi-Pico-6018818.html

1.4 USART

USART ist eine Synchrone Serielle Schnittstelle in Robo view werden zu Kommunikation die Sende- Empfangs- sowie die Taktleitung verwendet. Generell werden die empfangenen Bits in einen Schieberegister „eingeschoben“ und die zu sendenden analog dazu über einem Schieberegister Herausgeschoben. Nach dem ein Datenbyte 9 Bit sowie Start und Stop Byte empfangen wurden wier das Datenbyte (ohne die Start/Stop Bits) in die FIFI geschrieben. Die Erkennung das ein neuer Bit an den Eingang anliegt, wird über die Taktleitung getriggert. Das Senden der Daten funktioniert analog, nur werden hier bei einem entsprechenden Zustand des Taktpins die daten eingelesen.

1.5 Umsetzung

Dieses Projekt ist in Python umzusetzen. Dabei sollte die aktuelle Struktur des Projekts so weit wie möglich beibehalten werden. In der aktuellen Umsetzung werden Teile des Firmware über das ein- und ausblenden von Quelltexten konfiguriert um so z.b. Protokolle oder unterschiedliche Gerätetypen (Node, Device, Hub) zu implementieren. Für diese Vorgehensweise ist ein geeigneter Ersatz zu finden. Des weiteren ist die momentane lose Kopplung der Komponenten zu bewahren damit Später, bei Bedarf, auch andere Kommunikationsmedien wie z.B. SPI usw. zu Kommunikation verwendet werden können. Dies war bei dem ursprünglichen Projekt auch vorgesehen und entsprechend vorbereitet wurde aber nie umgesetzt. Um den Umfang dieser Arbeit nicht ausufern zu lassen, sollten nur die USART Schnittstelle, sowie die Verarbeitung der Systemnachrichten (siehe Unten) implementiert werden.

2 Kommunikationsframework

2.1 Übersicht

Aktuell ist das Kommunikationsframework in der Sprache C geschrieben. Einige Teile davon sind allerdings bereits in der PC Remoteanwendung auf Python portiert worden und können somit wiederverwendet werden. Dieses Framework ist für das Serialisieren der Pakete sowie deren Distribution zuständig. Das gesamte Kommunikationsnetz ist sternförmig aufgebaut. Es sind folgende Entitätentypen definiert:

- Data Hub – Hauptknoten des Roboters das zusätzlich noch mit der Hauptlogik verbunden ist. Die Logik kann über diese Entität mit anderen Entitäten des Roboters Komunizieren.
- Node – einerseits ein Nachrichtenendpunkt, aber auch ein Hub für an diese Node gegebenenfalls angeschlossene weitere Noden oder Endpunkte.
- Endpunkte – haben nur ein Ein- und Ausgang

2.2 Routing

Das Routing der Adressen und der Physikalischen Schnittstellen erfolgt über dynamische Adresstabellen – Noden sowie Endpunkte melden melden sich bei Start über einen „Ping“ Befehl bei dem Hub an und diese sowie alle Nodes die diese Nachricht weitervermittelt haben speichern die Kombination aus Adresse und Schnittstelle in internen Tabellen ab.

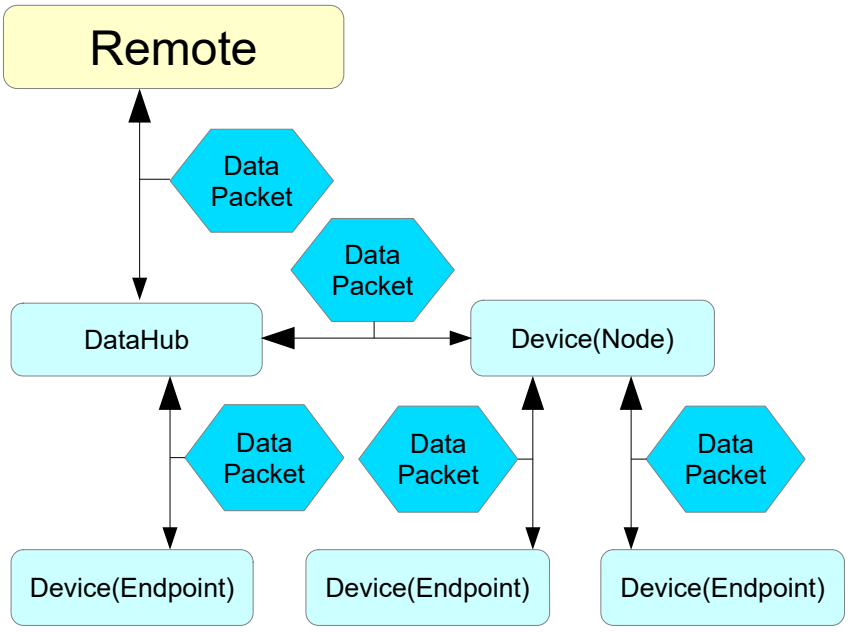


Abbildung 1: Beispiel einer Roboterstruktur in RoboControl

2.3 Datenpakete

Wie bereits erwähnt erfolgt der Datenaustausch über sogenannte Datenpakete. Diese bestehen aus einem Synchronisationszeichen, das auch gleichzeitig den Typ des Pakets kennzeichnet, einer Sendeadresse, einer Empfängeradresse, der Befehls-/Mesage-/Stream-ID, den Rohdaten sowie einem Endzeichen. Die tatsächliche Kodierung der Nachricht hängt von dem Verwendeten Datenformat momentan sind folgende Formate implementiert - 9Bit binär, 8Bit binär, 8Bit ASCII

- 9 Bit binär – das neunte Bit wird zu Synchronisation verwendet, alle Start sowie das Endzeichen haben ein gesetztes 9tes Bit. Somit können Rohdaten direkt unverändert übertragen werden. Dies ist das bevorzugteste Protokoll für die Kommunikation unter den Devices des Roboters.
- 8 Bit binär – hier wird das achte Bit zu Synchronisation verwendet, dadurch müssen die zu übertragenen Rohdaten uncodiert werden. Da nur sieben Datenbits in ein Datenbyte passen, werden die übriggebliebene Byte in dem folgenden Datenbyte übertragen. Um so eine Nachricht der Länge 5 Bytes zu übertragen sind somit 6 Bytes notwendig.
- 8 Bit ASCII – Daten werden als durch Menschen lesbare ASCII Zeichen versendet. Als Steuerzeichen werden nicht numerische ASCII Zeichen (z.b. „#“,“*“,“!“) verwendet. Die Datenbytes selber werden in ASCII Zahlen umgewandelt so wird das Datenbyte mit dem Wert 0x5A in die Zeichenkette „5A“ umgewandelt. Der Nachteil dieses Protokolls ist das für die Rohdaten die Doppelte Menge an Bytes zu übertragen ist. Dieses Protokoll wird momentan für das Fernverbindung mit dem PC verwendet. Für dieses Protokoll existiert bereits eine Python Implementierung.

2.4 Nachrichtentypen

Die Nachrichtentypen stellen eine Klassifikation der unterschiedlichen Funktionen eines Datenpakets:

1. Command – Als Command werden alle Befehle die an ein Device gesendet werden bezeichnet. Es können sowohl Stell werte als auch Requests sein.

- Message – Antwort auf ein Befehls
- Stream – Sind Periodisch übertragene Datenpakete mit Status bzw. Messwerten eines Devices
- Ok – Positive Quittierung eines Befehls
- Fail – Negative Quittierung eines Befehls
- Exception – Meldet eine in einem Device aufgetretene Ausnahme (wird momentan nicht verwendet)
- Allert – Meldet eine in einem Device aufgetretene Allarm (wird momentan nicht verwendet)

Die Ids der nachrichte sind 8 Bit groß, so ergeben sich in der momentanen umsetzung maximal 7 x 256 unterschiedliche Nachrichten pro Device.

Die ersten 32 IDs (0..32) in jeder Funktionsklasse sind für Systembefehle bzw. Nachrichten etc. reserviert. Die folgenden IDs (32..255) sind gerätespezifisch.

Die Systembefehle sind für alle Devices gleich so hat der ***Ping*** Befehl auf jedem Device immer die ID 0x03.

Die Zuordnung der gerätespezifischen IDs erfolgt nach dem Bedarfsprinzip. So kann es auch möglich sein das ein Device mehrere gleiche Nachrichten (z.b. ***get_value***) mit verschiedenen IDs verwendet die durch die unterschiedliche IDs die unterschiedlichen Komponenten eines Devices implementieren.

Aktuelle Repository für das C Framework

https://sourceforge.net/p/roboview/cframework/ci/master/tree/