Technische Universität Graz
Institut für Regelungs- und Automatisierungstechnik

Diplomarbeit

Synchrone Regelung von verteilten Servoantrieben am Beispiel eines Roboterfahrwerkes

---

Alexander Kalss
Graz, Austria, September 2010

Betreuer
Ao.Univ.-Prof. Dipl.-Ing. Dr.techn. Michael Hofbaur

Abstract

This diploma thesis was realized under the FWF project „Model based control of reconfigurable mobile robots“ (P20041-N15) and extends the previous work „Automation of reconfigurable multi-robot-systems“ (Institute of Automation and Control, TU-Graz). This thesis primary deals with synchronous control of several robot chassis, which can configured variable in its geometry and functionality.

In the theoretical part of this work we developed algorithms to calculate the kinematic robot model in a partly or a completely distributed way and to synchronously control the allocated wheel modules. For the realisation of the robot-controller we assumed that every wheel module of the robot chassis is configured with its own real-time-controller and contact sensors. With this configuration every chassis module is able to determine the geometry of the overall robot chassis through communication with the surrounding chassis modules. This provides the model for the robot geometry that is then used to set up the overall kinematics model for the mobile robot. This kinematic model represents the basis for the synchronous control of the robot's individual wheel actuators (brushless DC motors for steering and rotational wheel actuation). The implementation of our computational procedure supports variable configurations of the robot drive. In detail, we can handle single- and multi-robot systems with individually controlled robots and also completely distributed systems that realize single- and multi-robot systems. The practical part of this diploma thesis describes the implementation of the algorithm for synchronous and distributed control of mobile robots in various geometric configurations. It also deals with important implementation issues for distributed systems, such as time-synchronisation and communication among the distributed computational resources.

Tests with robots and multi-robot systems in various configurations document the real-world application of our work.

Kurzfassung

Diese Diplomarbeit wurde im Rahmen des FWF-Projektes (P20041-N15), aufbauend auf die vorangegangene Arbeit „Automatisierung von rekonfigurierbaren Multi-Roboter Systemen“, durchgeführt. Der Schwerpunkt dieser Diplomarbeit liegt in der synchronen Ansteuerung von mehreren Roboterfahrwerken, die in ihrer Geometrie und Funktionalität variabel gestaltet werden können.
Im Theorieteil dieser Arbeit werden Algorithmen zur teilweisen bzw. vollständig verteilten Berechnung der Roboterkinematik und zur verteilten synchronen Ansteuerung der Radmodule entwickelt. Für die Realisierung der Robotersteuerung wird angenommen, dass die einzelnen Radmodule des Fahrwerkes, welche bisher über ein angetriebenes und gelenktes Rad mit zugehöriger Aktuatorik und Leistungselektronik verfügten, nun auch jeweils mit einer eigenständigen Echtzeit-Recheneinheit und zusätzlichen Kontaktsensoren ausgestattet sind. Insbesondere die letztere Erweiterung ermöglicht den Fahrwerksmodulen, durch Kommunikation mit den Nachbarmodulen, die Geometrie des Gesamtfahrwerkes, als auch das daraus resultierende kinematische Modell des Gesamtfahrwerkes, verteilt zu berechnen. Dieses kinematische Fahrwerksmodell stellt die Basis für die synchrone Ansteuerung der einzelnen Radlenk- und Radantriebsmotoren (bürstenlose Gleichstrommotoren) der Radmodule dar. In der Implementation unterstützt der Algorithmus zur verteilten synchronen Ansteuerung der Radmodule auch variable Konfigurationen des Roboterfahrwerkes. So können Einzel- und Multi-Robotersysteme mit einzelnen zentral geregelten Robotern, sowie auch als (vollständig) verteilte Systeme realisierte Einzel- und Multi-Robotersysteme, angesteuert werden. Neben der Realisierung des beschriebenen Algorithmus für die Steuerung der Roboterfahrwerke wurde im praktischen Teil der Arbeit auch auf die für den Betrieb erforderliche inter-Modul bzw. inter-Roboter Kommunikation und die dafür notwendige Synchronisation der verteilten Recheneinheiten eingegangen.
Testfahrten mit unterschiedlich konfigurierten Roboterfahrwerken dokumentieren die praktische Realisierung der entwickelten Roboter-Fahrwerksregelung.

Danksagung

In erster Linie möchte ich mich bei Herrn Prof. Michael Hofbaur für die freundschaftliche Betreuung während der Verfassung dieser Diplomarbeit und für die finanzielle Förderung seitens des FWF-Projektes P20041-N15 recht herzlich bedanken.
Weiters gilt mein Dank auch der Forschungsholding TU Graz GmbH, welche durch den gewährten Microfund MF 02_08, den die für diese Diplomarbeit notwendigen Hardwareausbau der modularen Roboterplattform in ein Multi-Roboter System ermöglichte.
Besonderer Dank gebührt auch meinen Eltern, die mich stets in meinen Vorhaben bekräftigt haben und mich während meines Studiums auch finanziell unterstützt haben.
Schlussendlich möchte ich mich noch bei all meinen Studienkollegen und Freunden bedanken, die einen Großteil dazu beigetragen haben, dass die verlebte Studienzeit in Graz zu einem der bisherigen schönsten Lebensabschnitte wurde. Danke!

Alexander Kalss Graz, im September 2010

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

1.1 Mobile Roboter

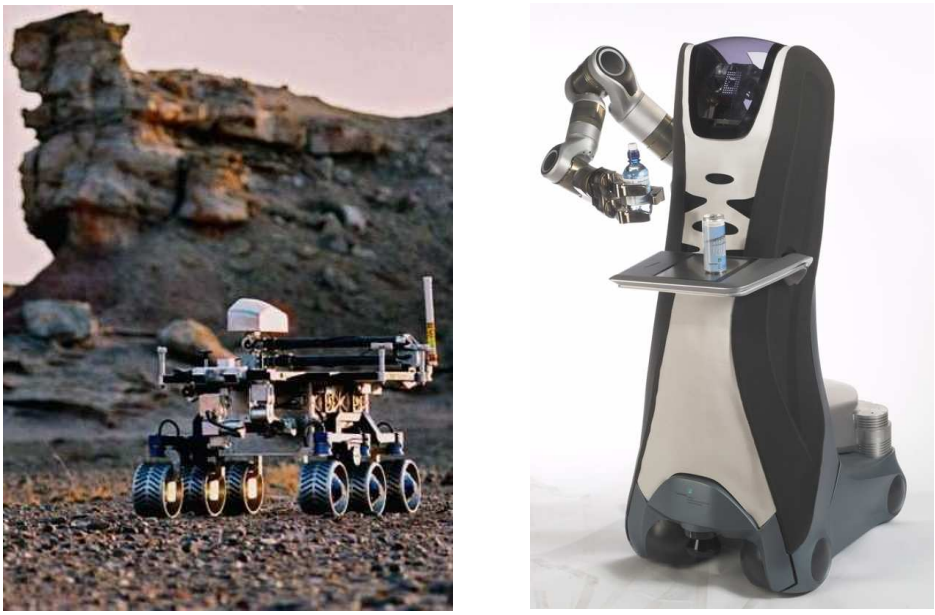
Immer leistungsstärkere Computersysteme in Kombination mit moderner Sensorik wie Laserscannern und ausgereiften Bildverarbeitungssystemen, bilden die Grundlage, das mobile Roboter mittlerweile in Einsatzbereiche vorgedrungen sind, die vor einigen Jahrzehnten noch unerdenklich waren.

(a) Erkundungsroboter

(b) Transportroboter

Abbildung 1.1: Mobile Robotersysteme 1

Im militärischen Bereich werden mobile Roboter beispielsweise für die Erkundung von feindlichen Gebieten oder für die Bergung und Entschärfung von Sprengmitteln eingesetzt. Im Logistikbereich werden mobile Roboter vermehrt im Teiletransport, zwischen den verschiedenen Fertigungsbereichen eines Unternehmens, eingesetzt.



(a) Forschungsroboter (b) Serviceroboter

Abbildung 1.2: Mobile Robotersysteme 2

Ein anderes Einsatzgebiet für mobile Roboter findet sich im Bereich der Weltraumforschung, speziell bei der Erforschung von erdnahen Planeten, wie zum Beispiel dem Mars. Aufgabe dieser autonomen Forschungsroboter ist es, die Oberfläche zu erkunden und auch mittels entsprechender Sensorik, Gesteinsproben vor Ort zu analysieren. Ein Vertreter dieser Art ist der Marsroboter „Spirit“, der im Auftrag der NASA entwickelt wurde und seit dem 4. Januar 2004, Untersuchungen auf der Marsoberfläche durchführt. Ein anderes Einsatzfeld indem die Bedeutung mobilen Roboter immer größer wird, ist der Haushalts- und Servicebereich. Als Beispiel können hier autonom agierende Bodenreinigungsroboter oder auch Roboter, die Lebensmittel servieren, genannt werden. Diese Serviceroboter sind jedoch noch Gegenstand aktiver Forschungsprojekte und finden derzeit im Haushaltsbereich noch wenig bis kaum Einsatz.

Je nach Anforderungs- und Einsatzgebiet sind diese mobilen Roboter mit unterschiedlichen Antriebssystemen ausgestattet. Hier kann bei mobilen Robotern großteils zwischen Ketten- und Radantrieben unterschieden werden. Bei Robotern deren Einsatzfeld sich in anspruchsvollem Gelände befindet, fällt die Wahl oft auf einen robusten Kettenantrieb. Bei mobilen Roboterfahrwerken, bei denen die Bewegungsfreiheit im Vordergrund steht,

kommen hauptsächlich gelenkte Radantriebe zum Einsatz. Bei gelenkten Rädern kann noch zwischen verschiedenen Radarten (Standardräder, Swedisch-Wheels,...) unterschieden werden.
Damit einem mobilen Roboter mitgeteilt werden kann, wo sich sein nächster anzufahrender Zielort befindet, kann den meisten mobilen Robotern ein Sollpfad vorgegeben werden. Bei modernen mobilen Robotersystemen erfolgt diese Pfadvorgabe nicht statisch, sondern kann vom mobilen Roboterfahrwerk je nach Situation adaptiert werden. Somit ist ein mobiler Roboter auch im Stande, den vorgegeben Sollpfad temporär zu verlassen, um einem Hindernis auszuweichen. Nachdem das Hindernis umfahren wurde, setzt der mobile Roboter die Fahrt am Sollpfad wieder fort.

1.1.1 Aufgaben und Ziele

Ziel dieser Diplomarbeit ist es, eine Multirobotersoftware für den Steuerungsprozess von modular und rekonfigurierbar aufgebauten Roboterfahrwerkssystemen zu entwickeln. Als Programmiersprache soll dabei LabVIEW von der Firma „National Instruments“[1] eingesetzt werden. Als theoretische Grundlage dienen dabei die in der Arbeit [14] ausgearbeiteten Berechnungsvorschriften für ein Multiroboterszenario. Neben der Steuerung eines Multiroboterfahrwerkes soll auch ein Algorithmus entwickelt werden, mit dem es möglich ist, das kinematische Modell eines Roboterfahrwerkes automatisch zu bestimmen. Abschließend werden die gesamten Aufgabenstellungen für diese Diplomarbeit nochmals kurz zusammengefasst:

- Der Entwurf eines Algorithmus, mit dem das kinematische Robotermodell automatisch über die Kontaktinformationen der einzelnen Robotermodule bestimmt werden kann.
- Die nähere Untersuchung und Ausarbeitung der Multiroboterkonfiguration „Vollständig verteilter Fall“.
- Mit Hilfe der Programmiersprache LabVIEW eine Multirobotersoftware zu entwickeln, mit der ein Multiroboterfahrwerk aufgrund eines vorgegebenen Sollpfades gesteuert werden kann.
- Abschließende Testfahrten um die entwickelte Multirobotersoftware an realen Multiroboterfahrwerken zu testen.

[1] www.ni.com

Kapitel 2

Hardware

2.1 Roboterfahrwerk

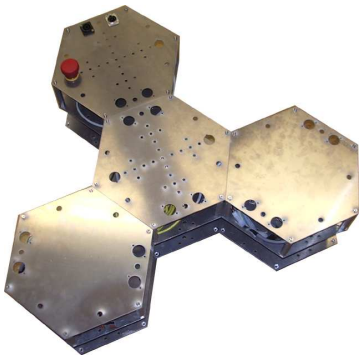


Abbildung 2.1: Roboterfahrwerk aus 4 Waben

Das Roboterfahrwerk wurde im Zuge einer Diplomarbeit [7] entwickelt und ist aus Aluminium-Platten und Fräskörpern aufgebaut. Ein Roboterfahrwerk entsteht durch den Zusammenschluss mehrerer Einzelwaben. Diese können über Verbindungsplatten miteinander verschraubt werden und bilden so ein stabiles Roboterfahrwerk. Durch einen Koppelmechanismus [13] besteht zudem die Möglichkeit, zwei oder mehrere unabhängige Roboterfahrwerke temporär miteinander zu verkoppeln. Bei den Einzelwaben gibt es verschiedene Wabentypen. Hier wären im speziellen Verbindungs-, Akku- oder CPU-Waben zu nennen. Die Verbindungswaben dienen rein zur Verbindung von anderen Wabentypen, wodurch Roboterfahrwerke mit unterschiedlichen Geometrien aufgebaut werden können. Die Akkuwabe versorgt das Roboterfahrwerk mit Energie und beinhalten in der derzeitigen Ausführung zwei in Serie geschaltete 12 Volt NiMH-Akkus. Die CPU-Wabe beherbergt

einen Industrie-PC vom Typ "Compact RIO“ der Firma „National Instruments“ [1] , der die Steuerung der einzelnen Radwaben des Roboterfahrwerkes übernimmt.

2.2 Interaktion der Hardware-Komponenten eines Roboterfahrwerkes

In Abbildung 2.2 sind schematisch die Hardware-Komponenten eines Roboterfahrwerkes, inklusive dem Windows-PC, dargestellt. Der Windows-PC dient als Bedien- und Anzeigeterminal für das Roboterfahrwerk und ist über WLAN (Wireless Local Area Network) mit dem Industrie-PC „Compact RIO“ des Roboterfahrwerkes, der abgekürzt auch als „cRIO“ bezeichnet wird, verbunden. Über den Windows-PC, der in weiterer Folge auch als „Host-PC“ bezeichnet wird, kann der Anwender einen gewünschten abzufahrenden Sollpfad für das Roboterfahrwerk vorgeben.

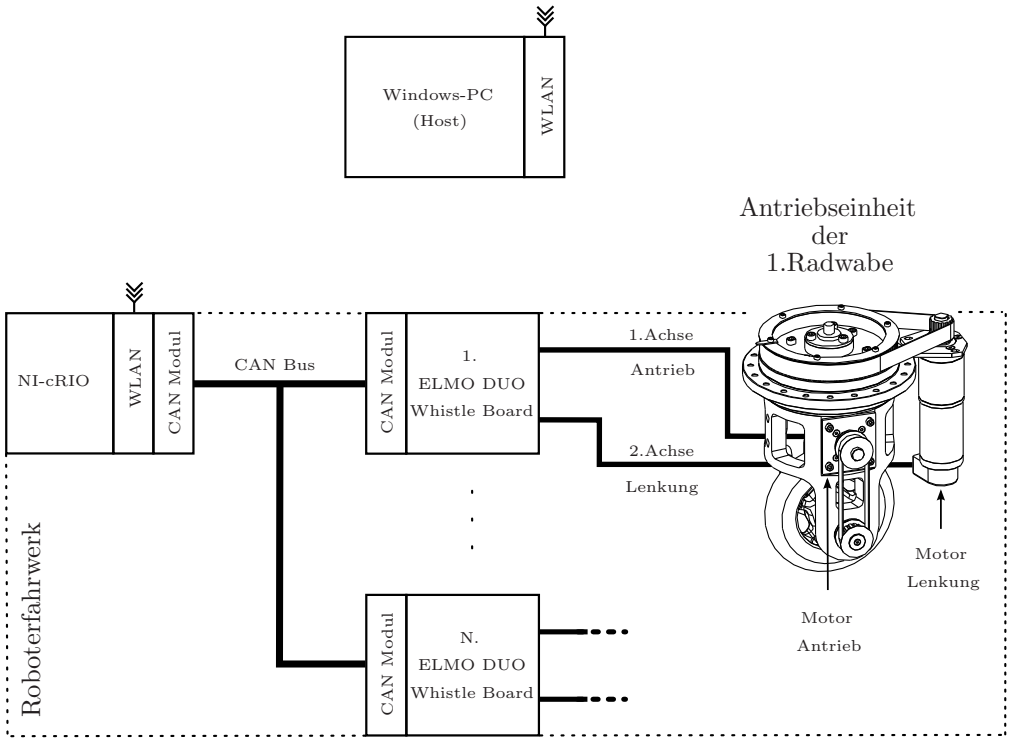


Abbildung 2.2: Schematische Verbindung der Hardware-Komponenten

Über das Bedienterminal, des am Host-PC ausgeführten Roboterprogramms, wird die Fahrt des Roboterfahrwerkes gestartet. Dabei sendet der Host-PC die Daten des Soll-

[1] www.ni.com/

pfades über ein WLAN-Netzwerk an den cRIO des Masterroboter-Fahrwerkes, der dann die notwendigen Raddrehzahl- und Radlenkwinkelsollwerte für die einzelnen Radwaben des Fahrwerkes berechnet. Nach Beendigung der Fahrt werden die Antriebsdaten (Raddrehzahl und Radlenkwinkel) aller Radwaben am Anzeigeterminal des Host-PCs dargestellt. Jede Radwabe besitzt für die Regelung der beiden bürstenlosen Gleichstrommotoren (BLDC) ein Motorregelgerät vom Typ „DUO Whistle Board“ der Firma „ELMO“. Das DUO-Whistle-Board besteht aus zwei Motorregelgeräten des Typs „Whistle“, wobei ein Whistle für die Ansteuerung eines BLDC-Motors benötigt wird. Die DUO-Whistle-Boards sind über CAN-Bus mit dem Industrie-PC (cRIO) verbunden. Über diese Bus-Verbindung werden die DUO-Whistle-Boards von der Recheneinheit (cRIO) parametriert und gesteuert.

2.3 Aufbau einer Radwabe

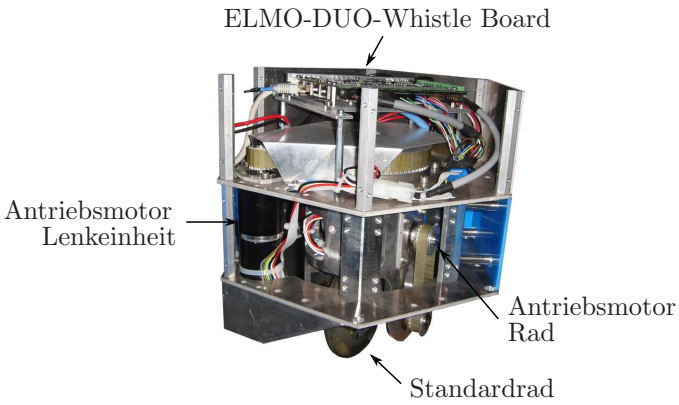


Abbildung 2.3: Radwabe mit Standardrad

Radwaben können mit unterschiedliche Radtypen aufgebaut werden (siehe Abb. 2.4). Das Standardrad (Abb. 2.4-1) ist jener Radtyp, der in einem drehbaren Käfig verbaut wird. Über zwei bürstenlose Gleichstrommotoren (siehe Punkt 2.3.1), welche über ein Motorregelgerät der Firma ELMO (siehe Punkt 2.3.2) angesteuert werden, wird das Rad angetrieben und gelenkt. Das omnidirektionale Rad oder Mechanum-Wheel (Abb. 2.4-2) muss nicht gelenkt werden. Bei einem Roboterfahrwerk mit diesem Radtyp ergeben sich durch unterschiedliche Raddrehzahlen andere Bewegungsrichtungen. Dadurch ist in diesem Fall der drehbare Käfig nicht notwendig. Das Schwenkrad oder Castor-Whell (Abb. 2.4-3) wird als unangetriebenes zusätzliches Stützrad verwendet und an Verbindungswaben verbaut. In den Abschnitten 2.3.1 bis 2.3.3 werden die elektrischen Komponenten einer

Radwabe noch genauer erläutert.

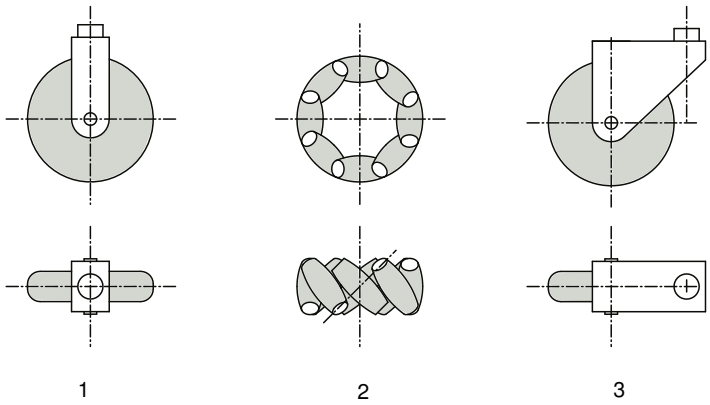


Abbildung 2.4: Übersicht der Radtypen

2.3.1 Antriebsmotor - BLDC(Maxon)

Bei den Antriebsmotoren einer Radwabe handelt es sich um bürstenlose Gleichstrommotoren (BLDC) vom Typ „EC$_{max}$ 40“ der Firma MAXON[2]. In Abbildung 2.5 ist ein solcher Motor mit Getriebe und Drehencoder dargestellt. Der Motor besitzt eine Antriebsleistung von 70 Watt und verfügt über ein aufgeflanschtes Planetengetriebe mit einem Übersetzungsverhältnis von 91/6 (≈15,166˙). Zusätzlich ist an der Motorwelle ein optischer Drehencoder montiert, der eine Umdrehung in 500 Inkremente auflösen kann. Durch eine entsprechende Signalauswertung des Encoders (4-fach-Auswertung) ist es schlussendlich möglich, eine Motorumdrehung in 2000 Inkremente aufzulösen.

Abbildung 2.5: BLDC EC$_{max}$ 40 (MAXON) mit Getriebe und Encoder

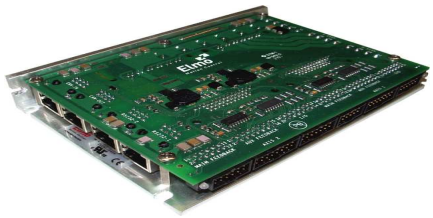


Abbildung 2.6: ELMO - DUO Whistle Board

2.3.2 Motorregelgerät - DUO Whistle Board(ELMO)

Das „DUO Whistle Board“ der Firma ELMO besteht aus zwei Motorregelgeräten des Typs „Whistle“ und ist zur Regelung von 2 Achsen ausgelegt. Das ELMO DUO-Whistle-Board verfügt über einen Spannungseingangsbereich von 7,5 bis 60 VDC, und ist in der Lage, einen Dauerstrom von 5 Ampere(effektiv) auf die einzelnen Motorwicklungen aufzuschalten. Nebenbei verfügt das Board über verschiedene Regelungsmodi wie Strom-, Drehzahl- und Positions-Regelung. Beim Roboterfahrwerk werden die ELMO-Whistles im Positionsregel-Modus betrieben. Um einen gleichmäßigen Bewegungsablauf der Radlenkeinheit zu erhalten, werden die von der Recheneinheit (cRIO) berechneten Radlenkwinkel-Sollwerte beziehungsweise die umgerechneten Soll-Positionswerte, nicht direkt dem Positionsregelkreis des ELMO-Whistles übergeben, sondern in eine „Position Time Table (PTT)“ auf dem ELMO-Whistle geschrieben. Die Position-Time-Table (PTT) ist ein Speicherbereich, in dem eine Sequenz von absoluten Positionswerten geschrieben werden kann. Nach dem Start der PTT wird diese Sequenz von Positionswerten schrittweise, beginnend mit dem ersten Soll-Positionswert, vom Motorregler zu den festgelegten Zeitpunkten angefahren. Der Vorteil durch die Verwendung der PTT liegt darin, dass das ELMO-Whistle zwischen den einzelnen absoluten Positionsvorgaben ein Interpolationspolynom 3. Ordnung berechnet. Dadurch wird ein sehr glatter Bewegungsablauf bei den Radlenk- und Radantriebsmotoren erzielt. Bei einer Radwabe regelt das erste Whistle des DUO-Whistle-Boards die Raddrehzahl und das zweite Whistle den Radlenkwinkel. Die von der Radwabe benötigte Lichtschranke, zur Referenzierung der Nullposition der Radlenkeinheit, wird über das zweite ELMO-Whistle eingelesen. Eine ausführliche Beschreibung zur Referenzierung der Radlenkeinheit ist in der Projektarbeit [8] zu finden.

Abbildung 2.7: Bildbeispiel NI-cRIO

2.3.3 Industrie PC - Compact RIO(NI)

Der „Compact RIO-9014“ von National Instruments[3] ist ein Industrie-PC mit einem Echtzeit-Betriebssystem und einem 400 MHz Pentium-Class Prozessor für deterministische Steuer- und Regelungsanwendungen. Er verfügt über einen nichtflüchtigen Compact-Flash-Speicher mit 2 GB, DRAM-Speicher mit 128 MB, einem 10/100MASE-T-Ethernet-Anschluss und einer USB-Schnittstelle. Zusätzlich ist der NI-cRIO mit einem CAN-Bus-Modul ausgestattet, das über zwei CAN-Bus-Schnittstellen verfügt. Die erste CAN-Bus-Schnittstelle wird vom Modul intern mit Spannung versorgt und deshalb auch für die Kommunikation mit den Motorregelgeräten (ELMO-Whistles) verwendet. Die zweite CAN-Bus-Schnittstelle muss über eine externe Spannungsquelle versorgt werden und bleibt im Falle des Roboterfahrwerkes ungenutzt. Der NI-cRIO hat einen Spannungseingangsbereich von 9 bis 36 VDC und wird über die Programmiersprache „LabVIEW“ von National Instruments programmiert. Eine kurze Einführung in diese Programmiersprache wird im Punkt 6.1 vorgenommen.

[2] www.maxonmotor.com
[3] www.ni.com

Kapitel 3

Berechnung des Robotermodells

Das kinematische Verhalten eines Rades kann mathematisch über seine Roll- und Gleitbedingung ausgedrückt werden. Dabei legt die Rollbedingung das kinematische Verhalten des Rades in Abrollrichtung und die Gleitbedingung das kinematische Verhalten des Rades senkrecht zur Abrollrichtung fest. Anhand der Roll- und Gleitbedingungen eines Rades kann ein Bewegungsraum für das Rad bestimmt werden, der Auskunft über die ausführbaren Bewegungen des Rades gibt. Für eine genaue Herleitung der Roll- und Gleitbedingungen wird auf [15] verwiesen. Eine ähnliche und allgemeinere Form zur Berechnung des vollständigen Bewegungsraumes (unabhängig vom spezifischen Radlenkwinkel β_i) kann über das in [9] angegebene Verfahren, der so genannten „Qualitativen Roll- und Gleitbedingungen“, durchgeführt werden. Diese qualitativen Roll- und Gleitbedingungen sind im Anhang A.2 für ein Standardrad angegeben. Da ein Roboterfahrwerk in der Regel aus mehreren Rädern besteht, können die qualitativen Roll- und Gleitbedingungen $\mathbf{j}_{q,i}$ und $\mathbf{c}_{q,i}$ der einzelnen Räder i, in den qualitativen Roll- und Gleitbedingungsmatrizen $\mathbf{J}_q$ und $\mathbf{C}_q$ (3.1) zusammengefasst werden.

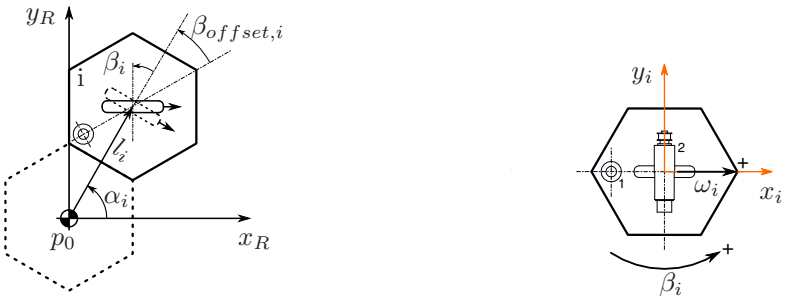
$$\mathbf{J}_q = \begin{bmatrix} \mathbf{j}_{q,1} \\ \mathbf{j}_{q,2} \\ \vdots \\ \mathbf{j}_{q,n} \end{bmatrix} \qquad \mathbf{C}_q = \begin{bmatrix} \mathbf{c}_{q,1} \\ \mathbf{c}_{q,2} \\ \vdots \\ \mathbf{c}_{q,n} \end{bmatrix} \tag{3.1}$$

n ... Gesamtanzahl der Radwaben pro Roboterfahrwerk.

Die beiden Matrizen $\mathbf{J}_q$ und $\mathbf{C}_q$ legen das kinematische Verhalten eines Roboterfahrwerkes fest und repräsentieren somit das kinematische Robotermodell. Über das Robotermo-

dell beziehungsweise die qualitativen Roll- und Gleitbedingungsmatrizen kann die Menge der zulässigen und (an-)steuerbaren Roboterbewegungen, in der Folge vektoriell definiert und als der (Geschwindigkeitsvektor-)Raum **B** bezeichnet, berechnet werden. Damit kann über eine mathematische Rang-Operation festgestellt werden, ob ein vorliegender Fahrbefehl (definiert durch einen 3-dimensionalen Geschwindigkeitsvektor) in diesem Raum liegt und somit vom Roboterfahrwerk ausgeführt werden kann. Die Berechnung des Raumes **B** und dessen Verwendung wird im Kapitel 4 Abschnitt 4.1.10 noch näher beschrieben. Um die qualitativen Roll- und Gleitbedingung eines Rades über die Tabelle A.2 bestimmen zu können, wird die Position des Rades in polaren Abstandskoordinaten, bezogen auf den Robotermittelpunkt p_0, benötigt (siehe Abbildung 3.1a). In der ersten Implementation der modell-basierten Roboterregelung [14] werden die polaren Abstandskoordinaten vom Anwender selbst in der Modelldefinition des entsprechenden Roboterfahrwerkes eingegeben. In dieser Modelldefinition des Roboterfahrwerkes sind Daten wie Position der Fahrwerksräder, Radtypen, Lenk-Offsetwinkel und die im CAN-Netzwerk verwendeten Radwaben-IDs gespeichert. Diese Informationen werden von der zentralen Recheneinheit am Beginn der Initialisierung ausgelesen, um daraus die qualitativen Roll- und Gleitbedingungsmatrizen $\mathbf{J}_q$ und $\mathbf{C}_q$ für das Roboterfahrwerk zu berechnen. Da die Abstandskoordinaten der einzelnen Radwaben bezogen auf den Robotermittelpunkt p_0 in dieser Modelldefinition statisch definiert sind, muss bei einer Änderung der Fahrwerkskonfiguration, zum Beispiel beim Hinzufügen einer zusätzlichen Radwabe zum Roboterfahrwerk, die Modelldefinition wieder manuell angepasst werden. Um diesen Schritt zu automatisieren und um autonom konfigurierbare Roboter- und Multirobotersysteme zu realisieren, wird im folgenden Abschnitt 3.2 eine Möglichkeit vorgestellt, wie das kinematische Robotermodell automatisch über die Kontaktinformationen der einzelnen Waben bestimmt werden kann. Voraussetzung hierfür ist, dass jede Wabe eine eigene Recheneinheit besitzt, über entsprechende Kontaktsensoren zur Erkennung von angedockten Waben verfügt und auch über die Möglichkeit verfügt, Verbindungsinformationen mit den Nachbarwaben austauschen zu können. Vorteil dieser automatischen Berechnung des Robotermodells ist, dass bei einer neuen oder geänderten Fahrwerkskonfiguration, die Modelldefinition vom Anwender nicht mehr eingegeben oder angepasst werden muss, sondern von den verteilten Recheneinheiten des Roboterfahrwerkes selbst bestimmt werden kann. Somit ist das Roboterfahrwerk in der Lage, sein Robotermodell selbst zu berechnen und diese Daten übergeordneten Programmteilen zur Verfügung zu stellen. In Punkt 3.1 wird kurz die Methode vorgestellt, mit der das Robotermodell über die polaren Abstandskoordinaten der Radwaben berech-

net wird. Der Punkt 3.2 befasst sich mit der Methode der autonomen Bestimmung des Robotermodells über die Kontaktinformationen der Waben.



(a) Abstandskoord. α_i und l_i und Lenkwinkel β_i einer Radwabe

(b) Festgelegte positive Fahrt- und Lenkrichtung ω_i und β_i

Abbildung 3.1: Abstandskoordinaten und positive Dreh- und Lenkrichtung einer Radwabe

3.1 Berechnung des Robotermodells mittels polarer Abstandskoordinaten

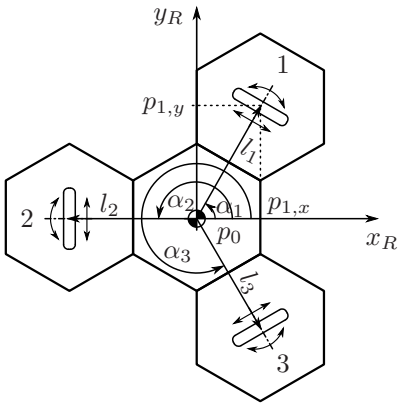
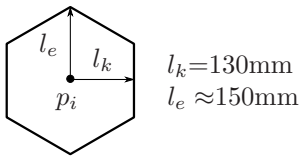


Abbildung 3.2: Robotermodell mit 3 aktiven Radwaben

Bei der Erstellung des Robotermodells über die polaren Abstandskoordinaten müssen die Winkel α_i und die Abstände l_i vom Robotermittelpunkt p_0 zu den einzelnen Radwabenmittelpunkte p_i (i=[1,..,n], n...Gesamtanzahl der Radwaben) bekannt sein (siehe Abbildung 3.2) beziehungsweise berechnet werden. Bei Rädern die nur angetrieben und nicht gelenkt werden, den so genannten „Fixed Wheels“, müssen auch noch die fest ein-

gestellten Radwinkel β_i bekannt sein. Eine Möglichkeit die Winkel α_i und die Abstände l_i zu berechnen ist, wenn die Positionen der Radwaben zuerst in kartesischen Koordinaten angegeben wird, da diese sehr einfach über die Abmessungen der Radwaben bestimmt werden können (siehe Abbildung 3.3). Der Abstand l_k vom Mittelpunkt einer Wabe bis zu einer Kante beträgt 130 mm, der Abstand l_e vom Mittelpunkt zu einer Ecke beträgt ungefähr 150 mm (siehe Gleichung 3.2). Um die x- und y-Abstandskomponente einer Radwabe bezogen auf den Robotermittelpunkt p_0 angeben zu können, werden die Vielfachen der Mittelpunkt-Ecke l_e beziehungsweise die Mittelpunkt-Kante Abstände l_k bestimmt, die in der x-Abstandskomponente $p_{i,x}$ beziehungsweise y-Abstandskomponente $p_{i,y}$ enthalten sind.

$$l_e = \frac{l_k}{\cos(30°)} = \frac{130mm}{\cos(30°)} = 150.11107mm \tag{3.2}$$



Wabe i

Abbildung 3.3: Geometrische Abmessungen einer Wabe

Bei dem Roboterfahrwerk von Abbildung 3.2 bekommt man für die Position von Radwabe 1, für die x- und y-Abstandskomponente $p_{1,x}$ und $p_{1,y}$ folgende Werte.

$$p_{1,x} = 1 \cdot l_k$$
$$p_{1,y} = 1.5 \cdot l_e$$

Allgemein bekommt man für die x-Komponente von p_i,

$$p_{i,x} = k \cdot l_k \tag{3.3}$$

(k...Anzahl der benötigten Mittelpunkt-Ecke Abstände l_e, ausgehend von p_0, bis die Höhe von $p_{i,x}$ erreicht wurde.)

und für die y-Komponente von p_i,

$$p_{i,y} = e \cdot l_e \tag{3.4}$$

(e...Anzahl der benötigten Mittelpunkt-Kante Abstände l_k, ausgehend von p_0, bis die Höhe von $p_{i,y}$ erreicht wurde.)

Nun muss die kartesische Positionsangabe von p_i noch in die polare Abstandsdarstellung umgerechnet werden. Das erfolgt über die Gleichungen 3.5 und Tabelle 3.1.

$$l_i = \sqrt{{p_{i,x}}^2 + {p_{i,y}}^2} \tag{3.5}$$

Nachdem alle Positionen der Radwaben p_1 bis p_n, bezogen auf den Robotermittelpunkt p_0,

$p_{i,x}$	$p_{i,y}$	α_i
+	+	$\arctan\left(\frac{y_{R,i}}{x_{R,i}}\right)$
-	+	$\pi + \arctan\left(\frac{y_{R,i}}{x_{R,i}}\right)$
+	-	$\arctan\left(\frac{y_{R,i}}{x_{R,i}}\right)$
-	-	$-\pi + \arctan\left(\frac{y_{R,i}}{x_{R,i}}\right)$

Tabelle 3.1: Bestimmung des Winkel α_i

in polaren Koordinaten vorliegen, können die Matrizen $\mathbf{J}_q$ und $\mathbf{C}_q$ (3.6) für die qualitativen Roll- und Gleitbedingungen über die Tabelle A.2 aufgestellt werden.

$$\mathbf{J}_q = \begin{bmatrix} j_{q,1} \\ j_{q,2} \\ \vdots \\ j_{q,m} \end{bmatrix}, \qquad \mathbf{C}_q = \begin{bmatrix} c_{q,1} \\ c_{q,2} \\ \vdots \\ c_{q,m} \end{bmatrix} \tag{3.6}$$

3.2 Autonome Bestimmung des Robotermodells über die Kontaktinformationen der Waben

Damit das Robotermodell über die Kontaktinformationen der einzelnen Waben eines Roboterfahrwerkes berechnet werden kann, müssen folgende Voraussetzungen erfüllt sein:

- Jede Wabe verfügt über eine eigene Recheneinheit.
- Jede Wabe besitzt eine eindeutige ID.
- Es wird im vorhinein eine Master- und eine Cockpitwabe festgelegt. (Durch die Cockpitwabe läuft die x-Achse des globalen xy-Roboterkoordinatensystems. Das Attribut „Cockpitwabe“ kann jeder Wabe des Roboterfahrwerkes verliehen werden. Durch sie kann die Lage des globalen Roboterkoordinatensystems gesteuert werden.)

- Jede Wabe muss über eine Kommunikationsverbindung zu den angedockten Nachbarwaben verfügen.

Wenn die oben angeführten Voraussetzungen erfüllt sind, dann kann der Algorithmus zur „Autonomen Bestimmung des Robotermodells über die Kontaktinformationen der Waben“ auf der Recheneinheit jeder Wabe ausgeführt werden. Diese Berechnungsvorschrift enthält folgende Abschnitte:

Am Beginn steht der mehrmalige Informationsaustausch mit den Nachbarwaben (3.2.3) bis jede Wabe eine Kontaktliste mit den Positionen der anderen Waben, bezogen auf ihr wabeneigenes xy-Koordinatensystems, besitzt. Mittels dieser Positionsdaten kann die Wabe den Robotermittelpunkt $\mathbf{p}_{i,0}$ (3.2.4) berechnen. Über die festgelegte Cockpitwabe, wird dann die Lage des globalen xy-Koordinatensystems (Abbildung 3.9) festgelegt. Im Anschluss ist jede Radwabe in der Lage, ihre polaren Abstandskoordinaten zum gemeinsamen Robotermittelpunkt $\mathbf{p}_0$, bezogen auf das globale xy-Koordinatensystem des Fahrwerkes, zu ermitteln. Nachdem nun jede Radwabe ihre polaren Abstandskoordinaten zum Robotermittelpunkt kennt, können die Radwaben ihre qualitativen Roll- und Gleitbedingungen $\mathbf{j}_q$ und $\mathbf{c}_q$ berechnen. Dieses Wabenmodell wird dann an die festgelegte Master-Wabe gesendet, welche die qualitativen Roll- und Gleitbedingungsmatrizen $\mathbf{J}_q$ und $\mathbf{C}_q$ für das gesamte Roboterfahrwerk aufstellt. Über dieses Robotermodell wird der Raum der zulässigen und steuerbaren Bewegungen (Raum $\mathbf{B}$) von der Master-Wabe berechnet (siehe 4.1.10). Die Master-Wabe ist dann in der Lage zu überprüfen, ob ein gewünschter Fahrbefehl in dem Raum $\mathbf{B}$, der zulässigen und steuerbaren Bewegungen, enthalten ist oder nicht.
Die Vorgangsweise wie bei einer Anordnung von Robotern bzw. Waben, die gegenseitig ihre Positionen bezogen auf ihr lokales Koordinatensystem kennen, der gemeinsame Robotermittelpunkt p_0 und die Roboter- bzw. Wabenpositionen bezogen auf das globale xy-Koordinatensystem der Anordnung bestimmt werden, wurde dem Artikel [6] „Decentralized formation control for small-scale robot teams with anonymity“ entnommen.

3.2.1 Beschreibung der Notation

n ... Gesamtanzahl der Waben die den Roboter bilden.
i ... Aktuelle Wabe, aus deren Sicht Abstände und Positionen berechnet werden.
$\mathbf{j}$ ... Übrigen Waben, die den Roboter bilden, jedoch ohne die Wabe i.

$i = [1,..,n]$ $(j(k) \in \mathbf{j}) \wedge (j(k) \neq i)$ $k = 1,..,n\text{-}1$

Beispiel:

$n = 4$;
$i = 2$;
$k = [1,..,n\text{-}1] = [1,2,3]$;
$\mathbf{j} = [j(k_1),..,j(k_{n-1})] = [j(1), j(2), j(3)] = [1,3,4]$;

3.2.1.1 Positionen

$\mathbf{p}_0$ ... Position des Robotermittelpunktes, bezogen auf das xy-Koordinatensystem von Wabe i.

$$\mathbf{p}_{0,i} = \mathbf{L}_i(p_0) = \begin{bmatrix} L_i[x_0] \\ L_i[y_0] \end{bmatrix} \tag{3.7}$$

$\mathbf{p}_L$ ... Position der Master-Wabe, bezogen auf das xy-Koordinatensystem von Wabe i.

$$\mathbf{p}_{L,i} = \mathbf{L}_i(p_L) = \begin{bmatrix} L_i[x_L] \\ L_i[y_L] \end{bmatrix} \tag{3.8}$$

$\mathbf{p}_C$ ... Position der Cockpit-Wabe, bezogen auf das xy-Koordinatensystem von Wabe i.

Über die Cockpit-Wabe kann die Lage des globalen Roboterkoordinatensystems xy_R bestimmt werden. Die x_R-Achse läuft durch den Robotermittelpunkt p_0 und durch den Mittelpunkt p_C der Cockpit-Wabe.

$$\mathbf{p}_{C,i} = \mathbf{L}_i(p_C) = \begin{bmatrix} L_i[x_C] \\ L_i[y_C] \end{bmatrix} \tag{3.9}$$

$\mathbf{p}_{j(k)}$ ... Position der übrigen Waben bezogen auf das xy-Koordinatensystem von Wabe i.

$$\mathbf{p}_{j(k),i} = \mathbf{L}_i(p_{j(k)}) = \begin{bmatrix} L_i[x_{j(k)}] \\ L_i[y_{j(k)}] \end{bmatrix} \tag{3.10}$$

Der Index k bezeichnet eine Wabe aus der Menge von $\mathbf{j}$. $(j(k) \in \mathbf{j})$

3.2.1.2 Abstände

Der Abstand oder die Distanz zwischen zwischen der Wabe i und einer Wabe $j(k)$ wird angegeben durch, $dist(p_i,p_{j(k)})$ und ist definiert durch,

$$dist(p_i, p_{j(k)}) = \left|\mathbf{p}_i - \mathbf{p}_{j(k)}\right| = \left|\mathbf{p}_{j(k)}\right| = \sqrt{L_i[x_{j(k)}]^2 + L_i[y_{j(k)}]^2} \tag{3.11}$$

$\mathbf{p}_i$ ist ein Nullvektor, weil die Abstände auf die Wabe i bezogen werden.

$$\mathbf{p}_i = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{3.12}$$

3.2.1.3 Winkel zwischen zwei Vektoren

Der Winkel zwischen zwischen zwei Vektoren $\vec{d}_1$ und $\vec{d}_2$ wird angegeben durch, $ang(\vec{d}_1,\vec{d}_2)$ und kann zum Beispiel über das Skalarprodukt bestimmt werden, wenn beide Vektoren in normierter Form vorliegen. Zusätzlich muss noch das Vorzeichen des Winkels bestimmt werden.

$$ang(\vec{d}_1, \vec{d}_2) = \arccos\left\langle \vec{d}_1, \vec{d}_2 \right\rangle \tag{3.13}$$

3.2.2 Kontaktstellen einer Wabe

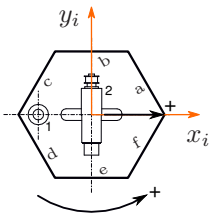


Abbildung 3.4: Wabe i mit den Kontaktstellen a-f

In Abbildung 3.4 ist eine einzelne Wabe mit dem fest verankerten lokalen xy-Wabenkoordinatensystem dargestellt. Da es sich bei der Wabe, wie schon aus der Bezeichnung ableitbar ist, um ein 6-Eck handelt, können maximal 6 Waben an eine einzelne Wabe direkt angedockt sein, wobei deren Position durch die Waben-Kontaktstelle a bis f eindeutig feststellbar ist.
In Abbildung 3.5 ist die Wabe 1 direkt mit der Waben 2 und Wabe 3 verbunden. Da das lokale xy-Wabenkoordinatensystem fest verankert ist und für alle Waben gleich definiert wurde(siehe Abbildung 3.4), hängt die Position der direkt angedockten Waben, bezogen

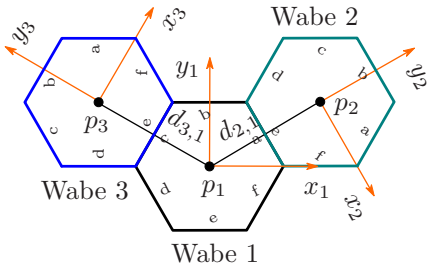


Abbildung 3.5: Zwei Waben in direkem Kontakt zu Wabe 1

auf das lokale Koordinatensystem von Wabe 1, nur mehr von der Kontaktstelle der Wabe 1 ab. In diesem Beispiel befindet sich der Mittelpunkt p_2 von Wabe 2 in einem Winkel von 30° und einer Länge von 0.26m zur Wabe 1. Die Winkel und Längen der 6 verschieden Positionen können der Tabelle 3.2 entnommen werden. Da alle Waben die selben Abmessungen besitzen, bleibt der Abstand zur Wabe 1 immer gleich, nur der Winkel ändern sich in Abhängigkeit der Kontaktstelle. Mittels Tabelle 3.2 sind die Positionen der direkt angedockten Waben definiert.

Wabe i	**Vektor** $d_{j(k),i}$	
Kontaktstelle	Winkel $\alpha_{j(k)}$	Länge $l_{j(k)}$
a	30°	0.26m
b	90°	0.26m
c	150°	0.26m
d	-150°	0.26m
e	-90°	0.26m
f	-30°	0.26m

Tabelle 3.2: Länge und Winkel von Vektor $d_{j(k),i}$ in Abhängigkeit der Kontaktstelle

Um die Position der indirekt angedockten Waben 3 in Abbildung 3.6 zu bestimmen, muss deren Position, angegebnen aus der Sicht des lokalen Koordinatensystems von Wabe 2, in das lokale Koordinatensystem von Wabe 1 umgerechnet werden. Das kann über eine Koordinaten-Transformation geschehen, wie sie in Gleichung 3.14 angegeben ist. Der Index j^*, i bedeutet, dass es sich um die Position der Wabe j^* bezogen auf das lokale Koordinatensystem von Wabe i handelt, deshalb $\mathbf{p}_{j^*,i}$.

$$\mathbf{p}_{j^*,i} = \mathbf{d}_{j^\#,i} + \mathbf{R}(\Theta) \cdot \mathbf{p}_{j^*,j^\#} \qquad (j^*, j^\# \in \mathbf{j}) \wedge (j^* \neq j^\#) \tag{3.14}$$

Die Rotationsmatrix $\mathbf{R}(\Theta)$ ist definiert durch,

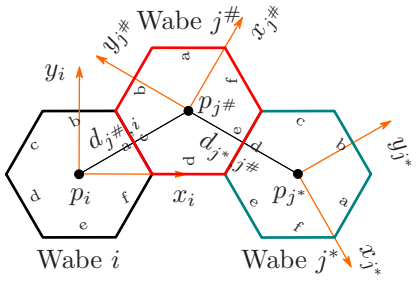


Abbildung 3.6: Indirekte Verbindung von Wabe j^* zu Wabe i

$$\mathbf{R}(\Theta) = \begin{bmatrix} \cos(\Theta) & -\sin(\Theta) \\ \sin(\Theta) & \cos(\Theta) \end{bmatrix} \tag{3.15}$$

Der Winkel Θ ist abhängig von der Lage der beiden lokalen Waben-Koordinatensystemen zueinander und kann über die Kontaktstellen bestimmt werden. Aus der Tabelle 3.3 kann der Winkel Θ entnommen werden. Wenn Wabe 1 mit der Kontaktfläche a an Wabe 2 und Wabe 2 mit der Kontaktfläche c an Wabe 1 andockt, dann ergibt sich ein Winkel Θ von 60° zwischen dem lokalen Koordinatensystem von Wabe 1 zu dem von Wabe 2. Über Gleichung 3.14 können Positionen, die sich auf das Koordinatensystem von Nachbarwaben beziehen, einfach in das wabeneigene xy-Koordinatensystem umgerechnet werden.

		Kontaktstelle Nachbarwabe $j^{\#}$					
	Θ	a	b	c	d	e	f
Wabe i	a	180	120	60	0	-60	-120
	b	-120	180	120	60	0	-60
	c	-60	-120	180	120	60	0
	d	0	-60	-120	180	120	60
	e	60	0	-60	-120	180	120
	f	120	60	0	-60	-120	180

Tabelle 3.3: Winkel Θ in Abhängigkeit der Kontaktstellen zueinander

Der Vektor $\mathbf{d}_{j^{\#},i}$ legt die Position der direkt angedockten Wabe $j^{\#}$ an die Wabe i fest und dessen polare Koordinaten können über die Tabelle 3.2 bestimmt werden. In Gleichung 3.16 ist der Vektor $\mathbf{d}_{j^{\#},i}$ in kartesischen Koordinaten angegeben.

$$\mathbf{d}_{j^{\#},i} = \begin{bmatrix} d_i[x_{j^{\#}}] \\ d_i[y_{j^{\#}}] \end{bmatrix} = \begin{bmatrix} \cos(\alpha_{j^{\#}}) \cdot l_{j^{\#}} \\ \sin(\alpha_{j^{\#}}) \cdot l_{j^{\#}} \end{bmatrix} \tag{3.16}$$

Der Punkt $\mathbf{p}_{j^*,j^\#}$ legt die Position der direkt angedockten Wabe j^* an Wabe $j^\#$ fest und dessen polare Koordinaten können gleich wie beim Vektor $\mathbf{d}_{j^\#,i}$ über die Tabelle 3.2 bestimmt werden. Somit ist Wabe i in der Lage, Positionsangaben, die sich auf ein lokales Koordinatensystem einer Nachbarwabe $j^\#$ beziehen, mittels Gleichung 3.14 in ihr wabeneignes lokales Koordinatensystem, umzurechnen.

3.2.3 Prinzip der Informationsweitergabe

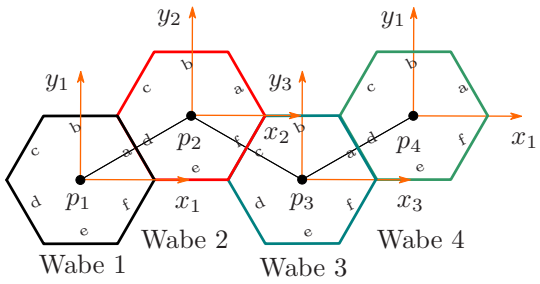


Abbildung 3.7: Zusammenschluß von 4 Waben

Jede Wabe i besitzt eine Kontakttabelle in der die IDs der direkt angedockten Nachbarwaben j, bezogen auf die Kontaktstellen a bis f, angegeben sind (siehe Tabelle 3.4). Zusätzlich sind noch Informationen über die Wabenart(Radwabe,Verbindungswabe,...) und den Rang der Wabe(Master- , Slave- oder Cockpitwabe) enthalten. Damit eine Wabe Informationen über ihre indirekten Kontakte bekommt, müssen diese von den direkt angedockten Nachbarwaben weiter geleitet werden. Für die Informationsweiterleitung müssen jedoch nur Waben sorgen, die mehr als eine Kontaktstelle aufweisen. Wabe 1 ist zum Beispiel nur direkt mit Wabe 2 verbunden und steht daher schon in der Kontakttabelle von Wabe 2. Wabe 1 kann daher keine Positionsinformationen von anderen Waben mehr in das System mit einbringen, deshalb wird Wabe 1 nur Positionsinformationen empfangen. Wabe 2 hingegen muss die Positionsinformation von Wabe 1 an Wabe 3 weiterleiten und diese wiederum an Wabe 4. In Abbildung 3.7 sind Wabe 2 und Wabe 3 für die Informationsweiterleitung verantwortlich. Am Ende dieser Informationsweiterleitung muss sichergestellt sein, dass Wabe 1 von Wabe 3 und 4 und das Wabe 4 von Wabe 1 und 2 die Positionsinformation besitzt.
Die Informationsweitergabe von Positionen funktioniert nach folgendem Schema: (Dieser Berechnungsalgorithmus läuft auf jeder Wabe)

Waben ID: 2 Kontaktstelle	ID Nachbarwabe	Kontaktstelle Nachbarwabe	Wabenart	Master	Cockpit-Wabe
a	-	-	-	-	-
b	-	-	-	-	-
c	-	-	-	-	-
d	1	a	-	0	0
e	-	-	-	-	-
f	3	c	-	0	0

Tabelle 3.4: Kontakttabelle zu Wabe 2 (siehe Abbildung 3.7)

1. Über die Kontakttabelle werden die direkten Kontakte berechnet und in eine Positionsliste mit zugehöriger ID geschrieben.

2. Hat die Wabe mehr als eine Kontaktstelle, dann setzt Sie mir Punkt 3 fort. Wenn nicht, dann Folgt Punkt 4.

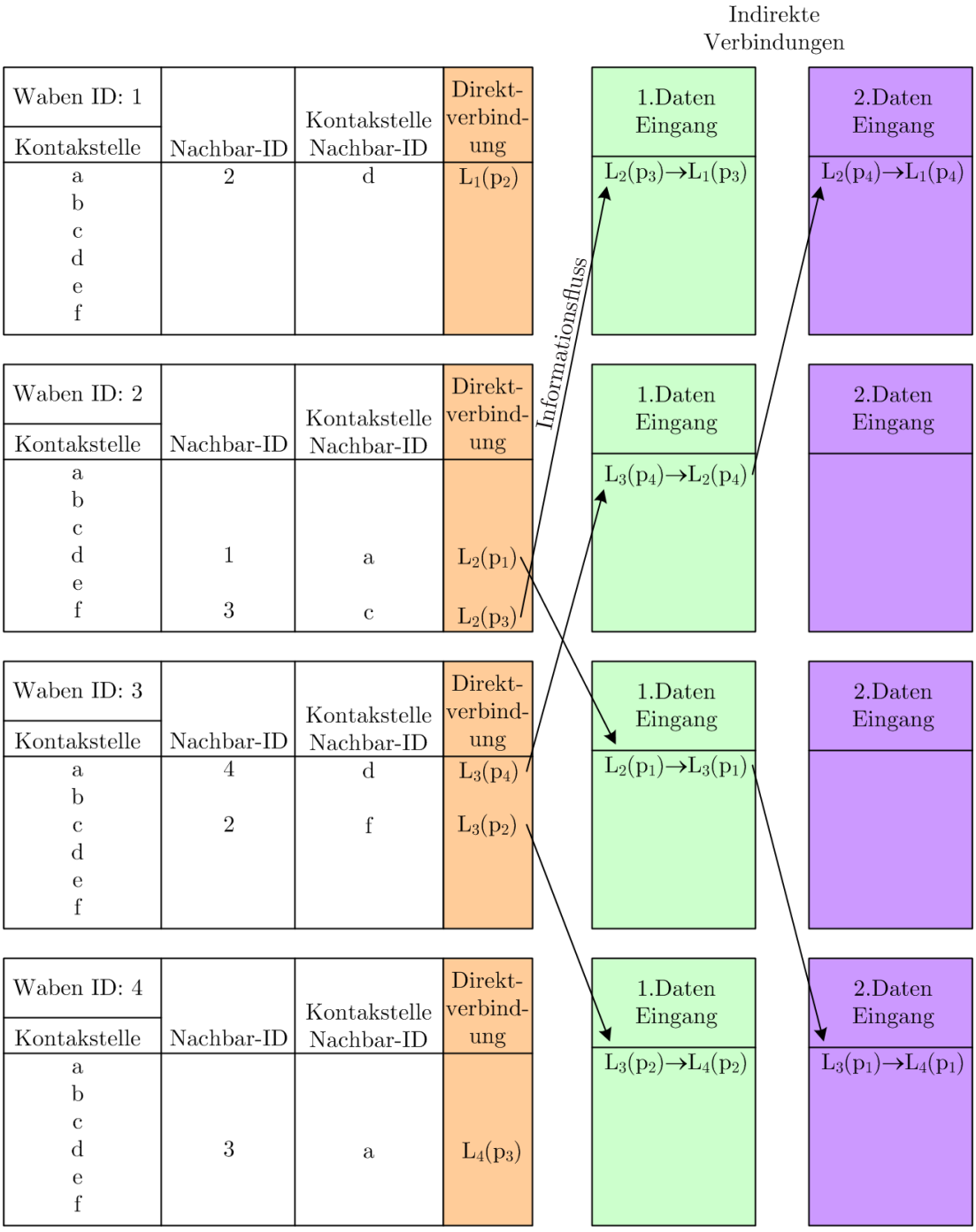
3. Als nächstes werden die direkten Kontaktinformationen an alle Nachbarwaben verteilt.

4. Warten auf Dateneingang. Wenn ein Dateneingang registriert wurde, wird mit Punkt 5 fortgefahren. Wenn das nicht der Fall ist, Abbruch des Algorithmus wenn das Abbruchkriterium erreicht wurde (z.B.: Counterlimit erreicht.)

5. Überprüfen ob die ID schon in der Positionsliste steht. Wenn nicht, weiter mit Punkt 6, ansonst weiter mit Punkt 4.

6. Die empfangen Positionsdaten werden über Gleichung 3.14 ins wabeneigene lokale Koordinatensystem umgerechnet und wieder an die Nachbarwaben verteilt, jedoch nicht mehr an jene Wabe von der man diese Positionsdaten erhalten hat. Weiter mit Punkt 4.

In Abbildung 3.8 ist der Informationsfluss für das Beispiel 3.7 graphisch dargestellt. Nachdem dieser Algorithmus auf jeder Wabe abgearbeitet wurde, besitzt jede Wabe die Positionsinformation aller anderen Waben, bezogen auf ihr eigenes lokales Koordinatensystem.



→ ... Umrechnen der Positionsinformation ins eigene lokale Wabenkoordinatensystem xy_i

Abbildung 3.8: Graphische Darstellung des Informationsflusses von Abbildung 3.7

3.2.4 Bestimmen von Abstand und Winkel zum gemeinsamen Robotermittelpunkt p_0

Nachdem alle Positionen der anderen Waben bestimmt wurden, die den Roboter bilden, kann nun der gemeinsame Robotermittelpunkt p_0 berechnet werden. Hierfür kann eine (n-1×2) Positionsmatrix $\mathbf{D}_i$ (Matrix 3.17) aufgestellt werden.

$$\mathbf{D}_i = \begin{bmatrix} L_i[x_{j(k_1)}] & L_i[y_{j(k_1)}] \\ L_i[x_{j(k_2)}] & L_i[y_{j(k_2)}] \\ \vdots & \vdots \\ L_i[x_{j(k_{n-1})}] & L_i[y_{j(k_{n-1})}] \end{bmatrix} \quad j(k) \neq i \quad i = 1, ..., n \tag{3.17}$$

Der individuelle Robotermittelpunkt $\mathbf{p}_{i,0}$ ergibt sich aus der vektoriellen Summer aller Abstände $\mathbf{p}_j$, gebrochen durch die Gesamtanzahl n der Waben.

$$\mathbf{p}_{0,i} = \begin{bmatrix} \frac{\sum_{k=1}^{n-1} L_i[x_{j(k)}]}{n} & \frac{\sum_{k=1}^{n-1} L_i[y_{j(k)}]}{n} \end{bmatrix}^T \tag{3.18}$$

Im nächsten Schritt wird die Position der Wabe bzw. der Abstand $\mathrm{x}_{R,i}$ und $\mathrm{y}_{R,i}$, bezogen auf das globale xy_R-Roboterkoordinatensystems berechnet, dessen x_R -Achse durch den Robotermittelpunkt $\mathbf{p}_{i,0}$ und durch die „Cockpit-Wabe“ geht. Die „Cockpit-Wabe“ wird im vorhinein definiert und ist jene Wabe, durch die die x_R -Achse des xy-Koordinatensystems gehen soll. Die x_R -Achse repräsentiert jene Richtung, in die sich der Roboter in positive Richtung fortbewegt. In vielen Fällen entspricht die Cockpit-Wabe auch der Master-Wabe, dass hängt von der Symmetrie des Roboters ab. Im Beispiel von Abbildung 3.9 wäre das auch möglich. Die x_R -Achse des globale xy_R-Roboterkoordinatensystems geht ebenfalls durch den Robotermittelpunkt $\mathbf{p}_{i,0}$, und steht senkrecht auf der x_R -Achse. In Abbildung 3.9 ist die Lage des xy_R-Koordinatensystems dargestellt.

$$x_{R,i} = dist(p_i, p_{0,i}) \cdot \cos\left(ang(\vec{x}_i, \vec{d}_i) - (ang(\vec{x}_i, \vec{x}_R)\right) \tag{3.19}$$

$$y_{R,i} = dist(p_i, p_{0,i}) \cdot \sin\left(ang(\vec{x}_i, \vec{d}_i) - (ang(\vec{x}_i, \vec{x}_R)\right) \tag{3.20}$$

Die Distanz zwischen dem Robotermittelpunkt $\mathbf{p}_{0,i}$ und dem Mittelpunkt $\mathbf{p}_i$ einer Wabe ist gegebnen durch

$$dist(p_i, p_0) = \sqrt{L_i[x_0]^2 + L_i[y_0]^2}\,. \tag{3.21}$$

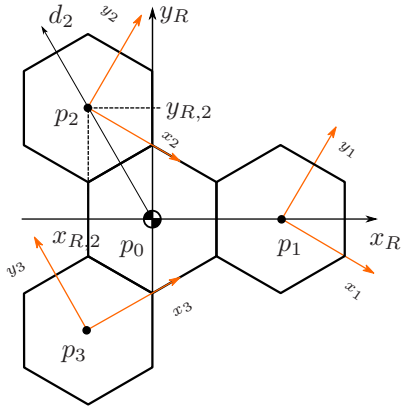


Abbildung 3.9: Lage des globalen xy_R-Koordinatensystems

Der Winkel zwischen $\mathbf{x}_i$ und $\mathbf{d}_i$ kann über das Skalarprodukt der beiden Vektoren bestimmt werden. Dieser Winkel wird aus Sicht des lokalen Waben-Koordinatensystems xy_i von Wabe i berechnet. Der Vektor $\vec{x}_i$ stellt einen Einheitsvektor dar, der in Richtung x-Achse zeigt. Dadurch ergibt sich für Vektor $\vec{x}_i$ folgende Form.

$$\mathbf{x}_i = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{3.22}$$

Die Richtung des Vektors $\vec{d_i}$ ist gegeben durch die Negation von $\mathbf{p}_{0,i}$, dadurch erhält man für den Vektor $\vec{d_i}$

$$\vec{d_i} = \mathbf{p}_{0,i} \cdot (-1) = \begin{bmatrix} -(L_i[x_0]) \\ -(L_i[y_0]) \end{bmatrix} . \tag{3.23}$$

Um das Skalarprodukt berechnen zu können, müssen die beiden Vektoren in normierter Form vorliegen. Bei Vektor $\vec{x}_i$ ist das, bedingt durch seine Definition, schon der Fall. Somit ergibt sich für den normierten Vektor $\vec{d}_{i,norm}$

$$\vec{d}_{i,norm} = \frac{1}{\left|\vec{d_i}\right|} \cdot \vec{d_i} = \frac{1}{\sqrt{{d_{i,x}}^2 + {d_{i,y}}^2}} \cdot \begin{bmatrix} d_{i,x} \\ d_{i,y} \end{bmatrix} . \tag{3.24}$$

Die Berechnung des Skalarproduktes ist gegeben durch

$$\left\langle \vec{x}_i, \vec{d}_{i,norm} \right\rangle = \frac{x_{i,x} \cdot d_{i,norm,x} + x_{i,y} \cdot d_{i,norm,y}}{\sqrt{{x_{i,x}}^2 \cdot {d_{i,norm,x}}^2} \cdot \sqrt{{x_{i,y}}^2 \cdot {d_{i,norm,y}}^2}} . \tag{3.25}$$

Damit erhält man für den Winkel zwischen Vektor $\vec{x}_i$ und $\vec{d_i}$

$$ang(\vec{x}_i, \vec{d_i}) = \arccos \left\langle \vec{x}_i, \vec{d}_{i,norm} \right\rangle . \tag{3.26}$$

Nun muss nur mehr das Vorzeichen des Winkels ang($\vec{x}_i$,$\vec{d_i}$) bestimmt werden, welches sich durch das Vorzeichen der y-Komponente von $\mathbf{p}_{0,i}$ ableiten lässt. (Siehe Tabelle 3.5) Der

$L_i[y_0]$	ang($\vec{x}_i$,$\vec{d_i}$)
≤ 0	positiv
> 0	negativ

Tabelle 3.5: Vorzeichen von $ang(\mathbf{x}_i, \mathbf{d}_i)$

Winkel zwischen dem Vektor $\vec{x}_i$ und $\vec{x}_R$ ist definiert durch

$$ang(\vec{x}_i, \vec{x}_R) = \arccos \left(\frac{L_i[x_C] - L_i[x_0]}{dist(p_{C,i}, p_{0,i})} \right) . \tag{3.27}$$

Die Distanz zwischen dem Robotermittelpunkt $\mathbf{p}_{0,i}$ und der Cockpit-Wabe $\mathbf{p}_{C,i}$ ist definiert durch

$$dist(p_{0,i}, p_{C,i}) = \left| \mathbf{p}_{C,i} - \mathbf{p}_{0,i} \right| . \tag{3.28}$$

Das Vorzeichen von ang($\vec{x}_i$,$\vec{x}_R$) kann über Tabelle 3.6 bestimmt werden. Aus den

	ang($\vec{x}_i$,$\vec{x}_R$)
$\mathrm{L}_i[\mathrm{y}_0] > \mathrm{L}_i[\mathrm{y}_C]$	negativ
$\mathrm{L}_i[\mathrm{y}_0] \leq \mathrm{L}_i[\mathrm{y}_C]$	positiv

Tabelle 3.6: Vorzeichen von $ang(\mathbf{x}_i, \mathbf{x}_R)$

Abständen $\mathrm{x}_{R,i}$ und $\mathrm{y}_{R,i}$ kann nun über die Tabelle 3.7 der Winkel α_i und über Gleichung 3.29 der Abstand l_i berechnet werden.

$\mathrm{x}_{R,i}$	$\mathrm{y}_{R,i}$	α_i
+	+	$\arctan\left(\frac{y_{R,i}}{x_{R,i}}\right)$
-	+	$\pi + \arctan\left(\frac{y_{R,i}}{x_{R,i}}\right)$
+	-	$\arctan\left(\frac{y_{R,i}}{x_{R,i}}\right)$
-	-	$-\pi + \arctan\left(\frac{y_{R,i}}{x_{R,i}}\right)$

Tabelle 3.7: Bestimmen des Winkel α_i

$$l_i = \sqrt{x_{R,i}{}^2 + y_{R,i}{}^2} \tag{3.29}$$

Da jetzt jede Radwabe ihre polaren Abstandskoordinaten (l_i,α_i) bezogen auf den gemeinsamen Robotermittelpunkt p_0 kennt, ist jede Radwabe in der Lage, ihre Matrizen $\mathbf{j}_{q,i}$ und $\mathbf{c}_{q,i}$ für die qualitativen Roll- und Gleitbedingungen (siehe A.2) aufzustellen.

3.3 Berechnungsbeispiele zur Erstellung des Robotermodells

Im Anschnitt 3.1 und 3.2 wurden zwei Methoden vorgestellt, wie das Robotermodell eines Roboterfahrwerkes bestimmt werden kann. Im Abschnitt 3.1 wurde das Robotermodell über die polaren Abstandskoordinaten der Radwaben zum Robotermittelpunkt p_0 bestimmt. Hiefür mussten im vorhinein vom Anwender selbst die kartesischen Abstandskoordinaten der Radwaben zum Robotermittelpunkt p_0 bestimmt werden, um daraus dann die polaren Abstandskoordinaten abzuleiten. Im Abschnitt 3.2 wurden die polaren Abstandskoordinaten der Radwaben, bezogen auf den Robotermittelpunkt p_0, automatisch über die Kontaktinformationen der einzelnen Waben gewonnen. Anhand dieser Daten wurden von der Master-Wabe die Roll- und Gleitbedingungsmatrizen $\mathbf{J}_q$ und $\mathbf{C}_q$ für das Roboterfahrwerk berechnet. Die Matrizen $\mathbf{J}_q$ und $\mathbf{C}_q$ repräsentieren das kinematische Robotermodell. Die beiden Berechnungsmethoden des Robotermodells sollen nun Anhand eines Roboterfahrwerkes mit 6 Waben (siehe Abbildung 3.10), davon 4 Radwaben, nochmals präsentiert werden.

3.3.1 Modellberechnung über die polare Abstandskoordinaten

Gegeben sei ein Roboterfahrwerk mit 4 Radwaben und festgelegtem globalen xy-Roboterkoordinatensystems (siehe Abbildung 3.10). Es soll nun über die polaren Abstandskoordinaten das Robotermodell beziehungsweise die Roll- und Gleitbedingungsmatrizen $\mathbf{J}_q$ und $\mathbf{C}_q$ des Roboterfahrwerkes bestimmt werden. Hierfür werden zuerst die kartesischen Abstandskoordinaten der Radwaben zum Robotermittelpunkt p_0 über die Gleichungen 3.3 und 3.4 bestimmt. Beispielhaft soll dies im folgenden für die Radwabe 1 geschehen.

$$p_{1,x} = 4 \cdot l_k = 2 \cdot 130mm = 260mm \tag{3.30}$$

$$p_{1,y} = 1,5 \cdot l_e = 1,5 \cdot 150mm = 225mm \tag{3.31}$$

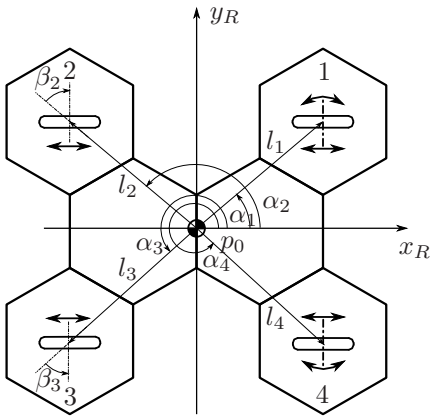


Abbildung 3.10: Roboterfahrwerk mit 4 Radwaben

Im Anschluss werden die kartesischen Abstandskoordinaten über die Gleichung 3.5 und die Tabelle 3.1 in polare Abstandskoordinaten umgerechnet, da diese für die Berechnung der qualitativen Roll- und Gleitbedingungs-Matrizen $\mathbf{j}_{q,i}$ und $\mathbf{c}_{q,i}$ benötigt werden. Für die Radwabe 1 werden folgende Werte ermittelt.

$$l_1 = \sqrt{{p_{1,x}}^2 + {p_{1,y}}^2} = \sqrt{260mm^2 + 225mm^2} = 344mm \tag{3.32}$$

$$\alpha_1 = \arctan\left(\frac{p_{1,y}}{p_{1,x}}\right) = \arctan\left(\frac{225mm}{260mm}\right) = 40,87^\circ \tag{3.33}$$

Die Positionsberechnung von Radwabe 2 bis 4 erfolgt in gleicher Weise, wie für Radwabe 1. In der Tabelle 3.8 sind die Abstandsdaten in polaren Koordinaten bezogen auf den Robotermittelkunkt p_0 für alle 4 Radwaben zusammengefasst. Als nächstes können die

Wabe i	Winkel [°]	Abstand [m]
Radwabe 1	$\alpha_1 = 40.87$	$l_1 = 0.344$
Radwabe 2	$\alpha_2 = 139.13$	$l_2 = 0.344$
Radwabe 3	$\alpha_3 = -139.13$	$l_3 = 0.344$
Radwabe 4	$\alpha_4 = -40.87$	$l_4 = 0.344$

Tabelle 3.8: Polare Abstandsdaten der Radwaben zum Robotermittelpunkt p_0(gerundet)

Matrizen für die Roll- und Gleitbedingungen $j_{q,i}$ und $c_{q,i}$ bestimmt werden. Die Berechnung erfolgt über Tabelle A.4. Die Ergebnisse für das Roboterfahrwerk aus Abbildung

3.10 sind in 3.34 angegeben.

$$\mathbf{J}_q = \begin{bmatrix} 0.3973 & -0.9176 & -0.5666 \\ 0.9176 & 0.3973 & 0 \\ 1 & 0 & -0.2251 \\ 1 & 0 & 0.2251 \\ -0.3973 & -0.9176 & -0.5666 \\ 0.9176 & -0.3973 & 0 \end{bmatrix}, \qquad \mathbf{C}_q = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.34}$$

3.3.2 Modellberechnung über die Kontaktinformation der einzelnen Waben

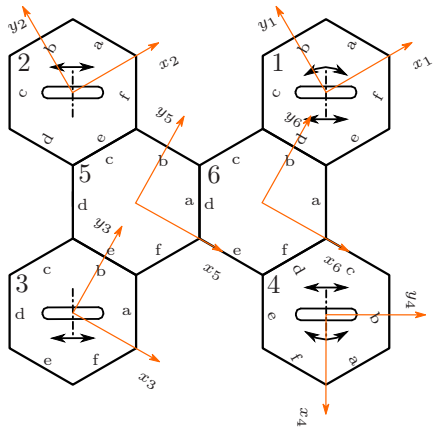


Abbildung 3.11: Beispielroboter mit 6 Waben

In Abbildung 3.11 ist vom Aufbau her ein identer Roboter dargestellt, wie im Beispiel von Abbildung 3.10. Jetzt soll das Robotermodell über die Kontaktinformationen der einzelnen Waben gewonnen werden. Hierfür muss für jede Wabe am Beginn eine Kontakttabelle vorliegen, in der die Verbindungen zu den unmittelbaren Nachbarwaben definiert sind. Auf die technische Umsetzung der Gewinnung dieser Kontakttabellen wird in dieser Arbeit verzichtet, da die theoretische Ausarbeitung im Vordergrund stehen soll. Damit der Informationsfluss zwischen der linker und rechten Seite (Wabe 2 und 3 zu Wabe 1 und 4) sowie auch entgegengesetzt sichergestellt werden kann, werden bei diesem Verfahren auch die Verbindungswaben (Wabe 5 und 6) benötigt. Ohne sie wäre der Informationsfluss zwischen den außen liegenden Waben unterbrochen. In der Tabelle 3.9 und 3.10 sind die Kontakttabellen beispielsweise für Wabe 1 und 6 dargestellt. Nachdem diese Kontaktinformationen zu den Nachbarwaben vorliegen, müssen diese auch an die anderen Waben verteilt werden. Das geschieht mit dem im Abschnitt 3.2.3 beschriebenen Verfahren der

Waben ID: 1 Kontaktstelle	ID Nachbarwabe	Kontaktstelle Nachbarwabe	Wabenart	Master	Cockpit-Wabe
a	-	-	-	-	-
b	-	-	-	-	-
c	-	-	-	-	-
d	6	b	Verbindungswabe	0	1
e	-	-	-	-	-
f	-	-	-	-	-

Tabelle 3.9: Kontakttabelle zu Wabe 1 (siehe Abbildung 3.11)

Waben ID: 6 Kontaktstelle	ID Nachbarwabe	Kontaktstelle Nachbarwabe	Wabenart	Master	Cockpit-Wabe
a	-	-	-	-	-
b	1	d	Radwabe	1	0
c	-	-	-	-	-
d	5	a	Verbindungswabe	0	0
e	-	-	-	-	-
f	4	d	Radwabe	0	0

Tabelle 3.10: Kontakttabelle zu Wabe 6 (siehe Abbildung 3.11)

Informationsweitergabe. Nachdem alle Kontaktinformationen ausgetauscht wurden, und in die lokalen Koordinatensysteme der einzelnen Waben umgerechnet wurden, besitzt jede Wabe die Positionsinformation der anderen Waben. In Tabelle 3.11 ist die Positionsliste von Wabe 1 angegeben. Mit den Positionsinformationen der anderen Waben kann nun der

Wabe i ID	$p_{i,x}$ [m]	$p_{i,y}$ [m]
Wabe 2	-0.450	0.260
Wabe 3	-0.675	-0.130
Wabe 4	-0.225	-0.390
Wabe 5	-0.450	0
Wabe 6	-0.225	-0.13

Tabelle 3.11: Positionsliste von Wabe 1

gemeinsame Robotermittelpunkt p_0 nach Gleichung 3.18 berechnet werden. Da für die Bestimmung des Robotermodells nur die Position der Radwaben von Bedeutung ist, sind die Positionsdaten der Verbindungswaben 5 und 6 nicht mehr von Interesse, deshalb scheinen sie in den weiteren Tabellen auch nicht mehr auf. Nach der Ermittlung des gemeinsamen Robotermittelpunktes p_0 und der Information der Lage der Cockpit-Wabe, kann nun das globale Roboterkoordinatensystem xy_R festgelegt werden. Wie schon im Abschnitt 3.2.4

Wabe i	p_{i,x_0} [m]	p_{i,y_0} [m]
Wabe 1	-0.337	-0.065
Wabe 2	0.113	-0.325
Wabe 3	0.113	0.325
Wabe 4	-0.225	-0.26

Tabelle 3.12: Robotermittelpunkt p_0 bezogen auf das lokale Koordinatensystem der 4 Radwaben

beschrieben wurde, geht die x_R-Achse durch denn Robotermittelpunkt p_0 und durch den Mittelpunkt p_c der Cockpit-Wabe, welche in diesem Beispiel die Wabe 6 ist. Die y_R-Achse geht ebenfalls durch den Robotermittelpunkt p_0, und steht senkrecht auf der x_R-Achse. In Abbildung 3.12 ist die Lage des globalen xy-Roboterkoordinatensystems dargestellt. Nach dem die Lage des globalen Roboterkoordinatensystems festgelegt wurde, können die

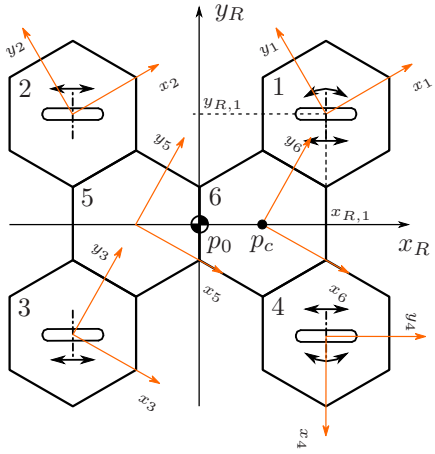


Abbildung 3.12: Lage des globalen xy-Roboterkoordinatensystems

Positionen der einzelnen Radwaben, bezogen auf das globale Roboterkoordinatensystem, berechnet werden. Dies geschieht mittels den Gleichungen 3.19 und 3.20. In der Tabelle 3.13 sind die Positionen der 4 Radwaben aufgelistet. Die kartesischen Positionsangaben aus

Wabe i	$x_{R,i}$ [m]	$y_{R,i}$ [m]
Wabe 1	0.260	0.225
Wabe 2	-0.260	0.225
Wabe 3	-0.260	-0.225
Wabe 4	0.260	-0.225

Tabelle 3.13: Position der 4 Radwaben bezogen auf das globale Roboter-Koordinatensystem

Tabelle 3.13 können über die Tabelle 3.7 und die Gleichung 3.29 in polare Abstandskoordinaten (Tabelle 3.14) umgerechnet werden. Nachdem die polaren Abstandskoordinaten der

Wabe i	α_i [°]	l_i [m]
Radwabe 1	$\alpha_1 = 40,87$	$l_1 = 0.344$
Radwabe 2	$\alpha_2 = 139,13$	$l_2 = 0.344$
Radwabe 3	$\alpha_3 = -139,13$	$l_3 = 0.344$
Radwabe 4	$\alpha_4 = -40,87$	$l_4 = 0.344$

Tabelle 3.14: Polare AbstandsKoordinaten der 4 Radwaben

4 Radwaben vorliegen, können wieder die qualitativen Roll- und Gleitbedingungsmatrizen $j_{q,i}$ und $c_{q,i}$ der einzelnen Räder über die Tabelle A.4 berechnet werden.

$$\mathbf{J}_q = \begin{bmatrix} 0.3973 & -0.9176 & -0.5666 \\ 0.9176 & 0.3973 & 0 \\ 1 & 0 & -0.2251 \\ 1 & 0 & 0.2251 \\ -0.3973 & -0.9176 & -0.5666 \\ 0.9176 & -0.3973 & 0 \end{bmatrix}, \qquad \mathbf{C}_q = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.35}$$

3.3.3 Fazit - Berechnungsmetoden des Robotermodells

Beide Berechnungsmethoden liefern die selben Ergebnisse, jedoch mit unterschiedlich großem Aufwand. Bei der Modellberechnung über die polaren Abstandskoordinaten (siehe Punkt 3.1) muss vom Anwender selbst sichergestellt werden, dass die Abstandskoordinaten der Radwaben zum Robotermittelpunkt p_0, die in der Modelldefinition gespeichert werden, für das vorliegende Roboterfahrwerk stimmen. Ändert sich die Radkonfiguration eines Roboterfahrwerkes, so muss das vom Anwender selbst in der Modelldefinition des Fahrwerkes berücksichtigt werden. Bei der automatischen Modellberechnung über die Kontaktinformationen der einzelnen Waben (siehe Punkt 3.2), braucht sich der Anwender um das Robotermodell beziehungsweise die Modelldefinition nicht mehr kümmern. Das Roboterfahrwerk kann autonom auf Änderungen in der Fahrwerkskonfiguration reagieren. Der Nachteil dieser Methode liegt im erhöhten Hardware-Aufwand. Jede Wabe benötigt eine Recheneinheit, Kontaktsensoren und eine Kommunikationsverbindung zu ihren Nachbarwaben. Schlussendlich muss vom Anwender selbst entschieden werden, ob eine autonome Erstellung des Roboterfahrwerkes notwendig ist oder nicht.

Kapitel 4

Multiroboter-Szenario

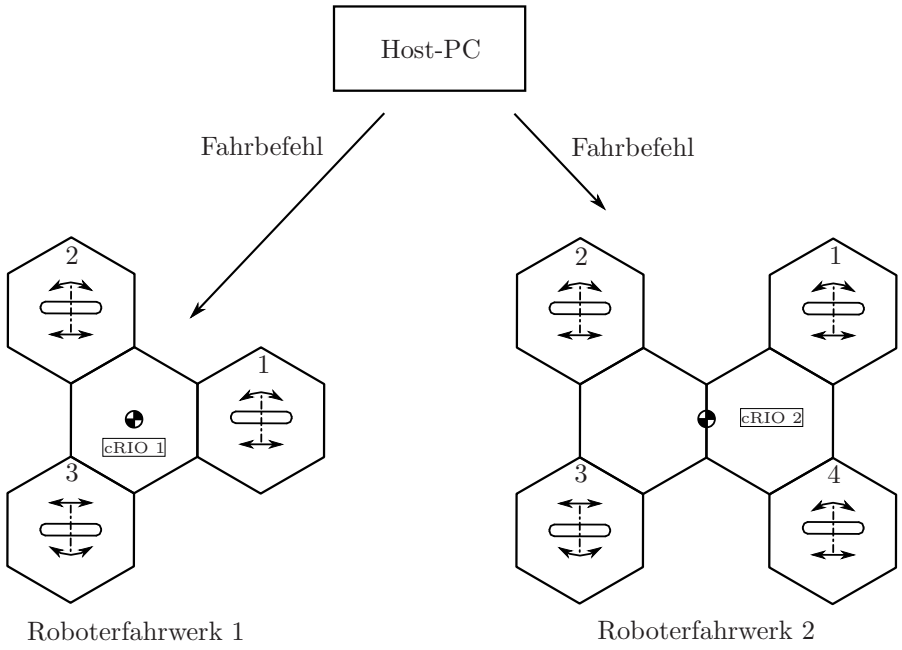


Abbildung 4.1: Multiroboter-Szenario - Konfigurationsbeispiel

Unter dem Multiroboter-Szenario wird die gleichzeitige Bewegung von mehrere Roboterfahrwerken starr, synchron oder variabel zueinander verstanden. „Starr zueinander“ bedeutet, dass zwei oder mehrere Roboterfahrwerke, zum Beispiel über einen Koppelmechanismus [13], fest miteinander verbunden sind. Dadurch entsteht ein zusammenhängendes Multi-Roboterfahrwerk, das aus mehreren eigenständigen Roboterfahrwerken aufgebaut ist. Eigenständig bedeutet in diesem Zusammenhang, dass das Roboterfahrwerk oder auch die Radwabe eine eigene Recheneinheit besitzt. Wenn sich die Roboterfahrwerke

„synchron zueinander“ bewegen, dann bewegen sich die einzelnen eigenständigen Fahrwerke in einer festgelegten Formation und halten dabei ihre Relativpositionen zueinander konstant. Unter einer „variablen Bewegung zueinander“ wird verstanden, dass sich die Roboterfahrwerke unabhängig voneinander bewegen. Dafür muss für jedes Fahrwerk ein individuell abzufahrender Sollpfad vorgegeben werden. Wenn sich die Roboterfahrwerke „starr“ oder „synchron“ zueinander bewegen, dann können die Sollpfade für die einzelnen Roboter aus dem Master-Pfad, der sich auf den definierten Mittelpunkt der Formation bezieht, abgeleitet werden. In Abbildung 4.1 ist ein Multiroboter-Szenario mit zwei eigenständigen Roboterfahrwerken dargestellt. Die beiden Fahrwerke besitzen jeweils eine Recheneinheit, sind aber aus unterschiedlich vielen Radwaben aufgebaut. Die maximale Anzahl der Radwaben aus denen ein einzelnes Roboterfahrwerk aufgebaut werden kann, ist aus theoretischer Sicht betrachtet nicht limitiert. In der praktischen Realisierung eines Roboterfahrwerkes sind jedoch limitierende Faktoren wie Busauslastung, Stromverbrauch, maximal zu vergebende IDs und einzuhaltende Zykluszeiten zu berücksichtigen. Wie im Kapitel 2.1 beschrieben wurde, bestehen die Roboterfahrwerke aus mehreren Waben unterschiedlicher Art. Die Radwaben können unter anderem Standardräder enthalten, die gelenkt und angetrieben werden können. Auch Schwenkräder kommen zum Einsatz. Diese werden hauptsächlich an Verbindungswaben befestigt und dienen zur Stabilisierung von Roboterfahrwerken, die aus weniger als drei Radwaben besteht.

4.1 Begriffsklärung

4.1.1 Indexierung im Multiroboterfall

n = Anzahl der Roboterfahrwerke
i = Ausgewählte Roboterfahrwerk $\qquad i = [1,..,n]$

m_i = Anzahl der Radwaben pro Roboterfahrwerk i
k_i = Ausgewählte Radwabe des Roboterfahrwerkes i $\qquad k_i = [1,..,m_i]$

Beispiele:

$\beta_{k_i,i} = \beta_{2,3} \Rightarrow$ Radlenkwinkel der 2.Radwabe vom Roboterfahrwerk 3
$\beta_{os(k_i,i)} = \beta_{1,2} \Rightarrow$ Radoffsetwinkel der 1.Radwabe vom Roboterfahrwerk 2
$\mathbf{p}_{0,i} = \mathbf{p}_{0,1} \Rightarrow$ Robotermittelpunkt $\mathbf{p}_0$ des 1.Roboterfahrwerkes

4.1.2 Definition der Roboter-Koordinatensysteme

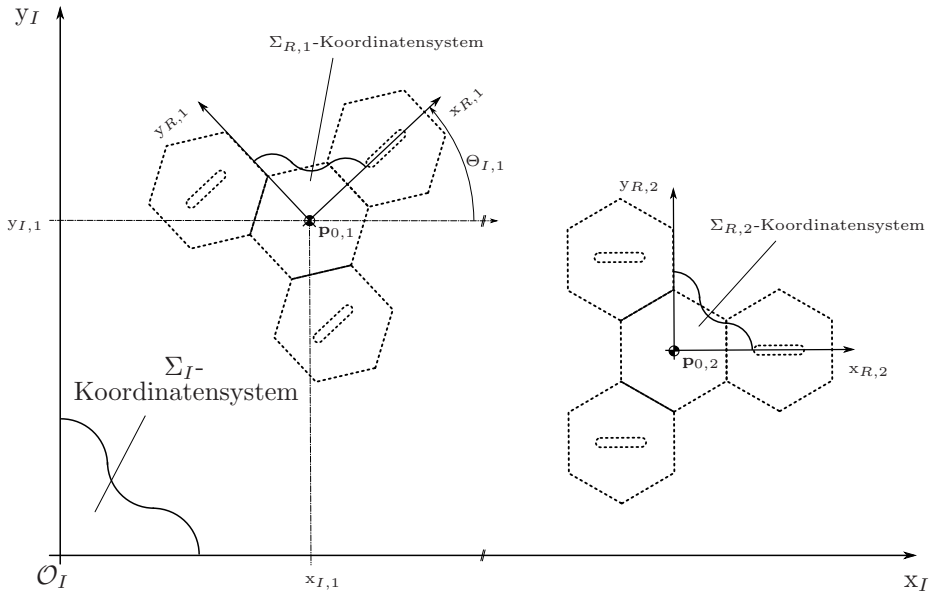


Abbildung 4.2: Roboter-Koordinatensysteme

In Abbildung 4.2 ist eine Multiroboteranordung mit den dazugehörigen xy-Koordinatensystemen dargestellt. Jedes Roboterfahrwerk i besitzt ein lokales xy-Koordinatensystem, dessen Ursprung sich im Robotermittelpunkt $\mathbf{p}_{0,i}$ befindet. Dieses lokale xy-Roboterkoordinatensystem wird in weiterer Folge als lokales $\Sigma_{R,i}$-Koordinatensystem bezeichnet. Damit die Positionen der einzelnen Roboterfahrwerke auf einen raumfesten Punkt $\mathcal{O}_I$ bezogen werden können, wird zusätzlich ein globales xy-Roboterkoordinatensystem eingeführt, das als Initial- oder Σ_I-Koordinatensystem bezeichnet wird. Über den globalen Lagevektor $\boldsymbol{\xi}_{I,i}$ (4.1),

$$\boldsymbol{\xi}_{I,i} = \begin{bmatrix} x_{I,i} \\ y_{I,i} \\ \Theta_{I,i} \end{bmatrix} \tag{4.1}$$

kann die Lage der Roboterfahrwerke i eindeutig in der 2 dimensionalen Bewegungsebnen bestimmt werden.

4.1.3 Host-PC

Der „Host-PC“, auch Hauptcomputer genannt, hat die Aufgabe, eine Mutiroboter-Anordnung zu steuern. Auf dem Host-PC wird vom Benutzer der abzufahrenden Sollpfad (siehe Punkt 4.1.4) für das Multiroboterfahrwerk vorgegeben. Er dient auch als Bedienterminal für den Steuerungsablauf. Das heißt, Kommandos wie „Fahrt starten“ oder „Fahrt beenden“ werden ebenfalls vom Host-PC an die beteiligten Roboterfahrwerke weitergeleitet. Neben der Steuerung des Multiroboterfahrwerks hat der Host-PC auch die Aufgabe, die eingelesenen Antriebs- (Raddrehzahlen und Radlenkwinkel) und Positionsdaten der einzelnen Roboterfahrwerke zu visualisieren.

4.1.4 Sollpfad

Der Sollpfad ist jener Pfad, der vom Multiroboterfahrwerk abgefahren werden soll und vom Benutzer am Host-PC vorgegeben wird. In Abhängigkeit der festgelegten Abtastzeit T_d der Recheneinheit und der vorgegebenen Fahrtgeschwindigkeit für das Roboterfahrwerk, werden am Host-PC aus dem Sollpfad die diskreten Fahrbefehle $\dot{\boldsymbol{\xi}}_k$ für das Roboterfahrwerk abgeleitet. Nachdem das Roboterfahrwerk alle diskreten Fahrbefehle $\dot{\boldsymbol{\xi}}_k$ ausgeführt hat, sollte es den vorgegeben Sollpfad abgefahren haben.

4.1.5 Modelldefinition

Die Modelldefinition legt fest, wie der spezifische Aufbau eines Roboterfahrwerkes aussieht. Darunter fallen unter anderem die Abstandparameter der Radwaben zum Robotermittelpunkt, die Radtypen (Standardrad, Omidirektionales Rad,...) und Raddurchmesser der Radwaben und sonstige Parameter, die die Konfiguration eines Roboterfahrwerkes eindeutig beschreiben. Die Modelldefinition liefert unter anderem die Grundlage für die Berechnung der Bewegungsräume (siehe Punkt 4.1.9) eines Roboterfahrwerkes.

4.1.6 Muliroboter-Mittelpunkt $\mathbf{p}_{0,m}$

Der am Host-PC festgelegte abzufahrende Sollpfad bezieht sich bei einem einzelnen Roboterfahrwerk immer nur auf einen Punkt, denn so genannten Robotermittelpunkt $\mathbf{p}_{0,i}$. Der Index i dient im Multiroboterfall dazu, die einzelnen Robotermittelpunkte der Fahrwerke zu unterscheiden. Der Robotermittelpunkt $\mathbf{p}_{0,i}$ legt dabei den Ursprung des lokalen Roboterkoordinatensystems $\Sigma_{R,i}$ fest. In Abbildung 4.3 ist der Robotermittelpunkt $\mathbf{p}_{0,1}$ des Roboterfahrwerkes 1 mit dem lokalen Roboterkoordinatensystem $\Sigma_{R,1}$ darge-

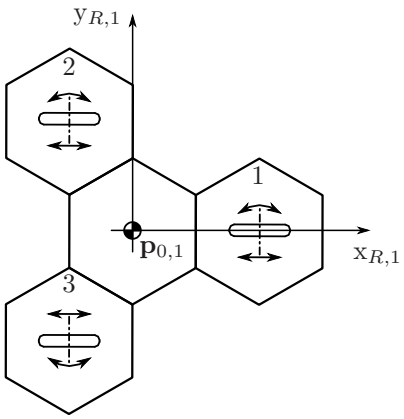


Abbildung 4.3: Festlegung des Robotermittelpunktes $\mathbf{p}_{0,1}$

stellt. Werden mehrere Roboterfahrwerke miteinander verbunden oder sollen sich mehrere Roboterfahrwerke in einer Formation bewegen, dann muss für diese Anordnung ein gemeinsamer Robotermittelpunkt, der sogennante Multi-Robotermittelpunkt $\mathbf{p}_{0,m}$, bestimmt werden. Der am Host-PC definierte Sollpfad bezieht sich bei einer Multiroboteranordung auf den festgelegten Multi-Robotermittelpunkt $\mathbf{p}_{0,m}$. In Abbildung 4.4 ist ein Multi-Roboterfahrwerk bestehend aus zwei zusammen gedockten Roboterfahrwerken mit dem Multi-Robotermittelpunkt $\mathbf{p}_{0,m}$ und dem lokalen Multiroboter-Koordinatensystem $\Sigma_{R,m}$ dargestellt.

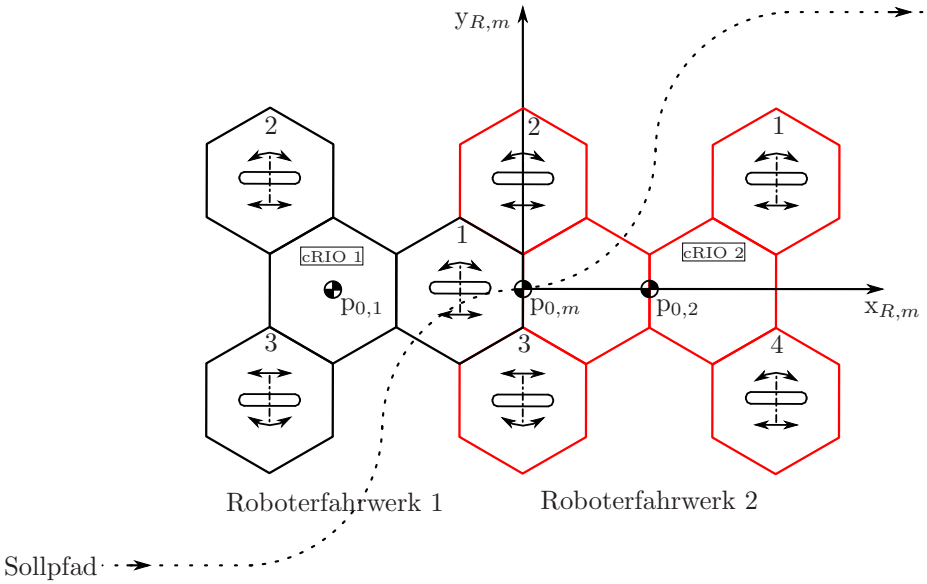


Abbildung 4.4: Festlegung des Multiroboter-Mittelpunktes $\mathbf{p}_{0,m}$

4.1.7 Bewegung in der Ebene

Im Folgenden werden 2 Möglichkeiten vorgestellt, wie die Bewegung eines Körpers in der Ebene mathematisch beschrieben werden kann. Die erste Möglichkeit geht über den Geschwindigkeitsvektor $\dot{\boldsymbol{\xi}}$ und die zweite Möglichkeit über den Momentanpol $\mathbf{p}_{MP}$ des Körpers. In diesem Abschnitt soll ein grundlegendes Verständnis über den Geschwindigkeitsvektor $\dot{\boldsymbol{\xi}}$ und den Momentanpol $\mathbf{p}_{MP}$ erzeugt werden, eine ausführliche Herleitung ist in [14] (Kapitel 2.3) beschrieben.

4.1.7.1 Fahrbefehl $\dot{\xi}$

Die Lage des lokalen Roboterkoordinatensystems Σ_R, bezogen auf das globale Roboterkoordinatensystems Σ_I, kann über den homogenen Lagevektor,

$$\boldsymbol{\xi_I} = \begin{bmatrix} x \\ y \\ \Theta \end{bmatrix} \tag{4.2}$$

eindeutig beschrieben werden. Leitet man den Lagevektor $\boldsymbol{\xi}_I$ zeitlich ab, erhält man den globalen Geschwindigkeitsvektor $\dot{\boldsymbol{\xi}}_I$.

$$\dot{\boldsymbol{\xi}}_I = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\Theta} \end{bmatrix} \tag{4.3}$$

Mittels dieses globalen Geschwindigkeitsvektoren $\dot{\boldsymbol{\xi}}_I$ kann die Bewegung des Roboterfahrwerkes, bezogen auf das globale Roboterkoordinatensystem Σ_I, eindeutig beschrieben werden. Um den globalen Geschwindigkeitsvektor $\dot{\boldsymbol{\xi}}_I$ in das lokale Koordinatensystem Σ_R des Roboterfahrwerkes zu transformieren, muss der globale Geschwindigkeitsvektor $\dot{\boldsymbol{\xi}}_I$ mit der Rotationsmatrix $\mathbf{R}(\Theta)$ (4.5) multipliziert werden. In Gleichung 4.4 ist diese Transformationsvorschrift angegeben.

$$\dot{\boldsymbol{\xi}}_R = \mathbf{R}(\Theta) \cdot \dot{\boldsymbol{\xi}}_I \tag{4.4}$$

$$\mathbf{R}(\Theta) = \begin{bmatrix} \cos(\Theta) & \sin(\Theta) & 0 \\ -\sin(\Theta) & \cos(\Theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.5}$$

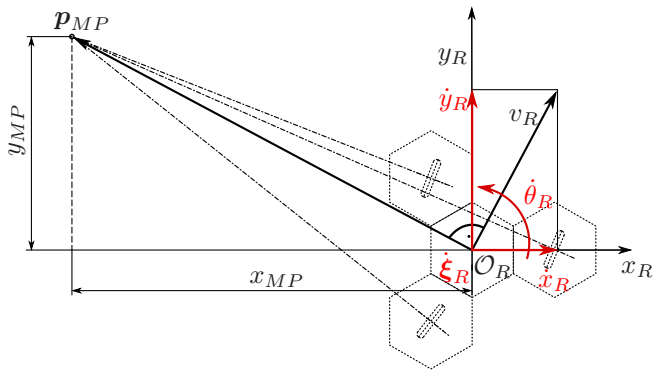


Abbildung 4.5: Definition des Momentanpols $\boldsymbol{p}_{MP}$

4.1.7.2 Momentanpol $\mathbf{p}_{MP}$

Jede durch einen Geschwindigkeitsvektor $\dot{\boldsymbol{\xi}}$ hervorgerufene Bewegung eines Körpers, kann als Drehung um einen bestimmten Punkt $\mathbf{p}_{MP}$, dem sogennanten Momentanpol, angesehen werden. Im lokalen Roboterkoordinatensystem Σ_R lässt sich immer ein Punkt finden, der keine translatorische Geschwindigkeit gegenüber dem globalen Roboterkoordinatensystem Σ_I besitzt. Ein Sonderfall ist die geradlinige Bewegung ($\dot{\Theta}_R$=0) des Roboterfahrwerkes. In diesem Fall liegt der Momentanpol $\mathbf{p}_{MP}$ im Unendlichen und wird als Fernpunkt bezeichnet. Bei einer Bewegung um den Momentanpol $\mathbf{p}_{MP}$ handelt es sich immer um eine reine Drehbewegung ohne Translation. In Abbildung 4.5 ist die Lage des Momentanpols $\mathbf{p}_{MP}$ für ein Roboterfahrwerk mit dem dazugehörigen lokalen Geschwindigkeitsvektor $\dot{\boldsymbol{\xi}}_R$ dargestellt. Der Momentanpol $\mathbf{p}_{MP}$ kann aus dem lokalen Geschwindigkeitsvektor $\dot{\boldsymbol{\xi}}_R$ des Roboterfahrwerkes berechnet werden. Dabei müssen zwei Fälle unterschieden werden, ob der Momentanpol $\mathbf{p}_{MP}$ im Endlichen oder im Unendlichen liegt. Wenn $\dot{\Theta}_R \neq 0$ ist, dann liegt der Momentanpol $\mathbf{p}_{MP}$ im Endlichen, und kann über Gleichung 4.6 berechnet werden.

$$\mathbf{p}_{MP} = \begin{bmatrix} \frac{-\dot{y}_R}{\dot{\Theta}_R} \\ \frac{\dot{x}_R}{\dot{\Theta}_R} \\ 1 \end{bmatrix} \tag{4.6}$$

Wenn $\dot{\Theta}_R$=0 ist, dann liegt der Momentanpol im Unendlichen. Der Richtungunksvektor $\mathbf{p}_{MP}$, zum im Unendlichen liegenden Momentanpol, kann über Gleichung 4.7 bestimmt werden.

$$\mathbf{p}_{MP} = \begin{bmatrix} -\dot{y}_R \\ \dot{x}_R \\ 0 \end{bmatrix} \tag{4.7}$$

4.1.8 Virtuelle Verschiebung des Robotermittelpunktes

Wie im Punkt 4.1.6 „Muliroboter-Mittelpunkt $\mathbf{p}_{0,m}$“ beschrieben wurde, bezieht sich der Fahrbefehl $\dot{\boldsymbol{\xi}}_i$ für ein Roboterfahrwerk i immer auf den Robotermittelpunkt $\mathbf{p}_{0,i}$. Wird aus mehreren Roboterfahrwerken ein Multiroboterfahrwerk mit zugehörigem Multiroboter-Mittelpunkt $\mathbf{p}_{0,m}$ gebildet, dann gibt es zwei Varianten, wie die lokalen Robotermittelpunkte $\mathbf{p}_{0,i}$ in den Multiroboter-Mittelpunkt $\mathbf{p}_{0,m}$ verschoben werden können. Bei der ersten Variante müssen die Modelldefinitionen (siehe Punkt 4.1.5) der Roboterfahrwerke so verändert werden, dass die lokalen Robotermittelpunkte $\mathbf{p}_{0,i}$ im Multiroboter-Mittelpunkt $\mathbf{p}_{0,m}$ zu liegen kommen. Zusätzlich müssen bei dieser Variante die Offsetwinkel $\beta_{os(k_i,i)}$ der einzelnen Fahrwerksräder neue berechnet werden. Die zweite Variante nennt sich „Virtuelle Verschiebung des Robotermittelpunktes“, bei der virtuell die lokalen Robotermittelpunkte $\mathbf{p}_{0,i}$ in den Multiroboter-Mittelpunkt $\mathbf{p}_{0,m}$ verschoben werden. Virtuell bedeutet, dass in Wirklichkeit der Multiroboter-Fahrbefehl $\dot{\boldsymbol{\xi}}_m$ in die einzelnen Robotermittelpunkte $\mathbf{p}_{0,i}$ transformiert wird. Bevor der Fahrbefehl $\dot{\boldsymbol{\xi}}_m$ in einen lokalen Robotermittelpunkt $\mathbf{p}_{0,i}$ transformiert werden kann, müssen die Transformationsparameter festgelegt werden. Das ist zum einen die Lage des Roboter-Mittelpunktes $\mathbf{p}_{0,i}$, aus Sicht des lokalen Multiroboter-Koordinatensystems $\Sigma_{R,m}$, der über die Parameter (siehe Abbildung 4.6) $x_{r,i}$ und $y_{r,i}$ definiert wird, und zum anderen der Winkel γ_i, der eine mögliche Verdrehung des lokalen Roboterkoordinatensystems $\Sigma_{R,i}$ gegenüber dem lokalen Multiroboter-Koordinatensystems $\Sigma_{R,m}$ beschreibt. Diese Transformationsparameter für einen Multiroboter-Fahrbefehl $\dot{\boldsymbol{\xi}}_m$ in die einzelen Robotermittelpunkte $\mathbf{p}_{0,i}$ werden immer aus Sicht des lokalen Multiroboter-Koordinatensystems $\Sigma_{R,m}$ angegeben und müssen für jedes Roboterfahrwerk i bestimmt werden, die Teil des Multiroboterfahrwerkes sind. In Abbildung 4.6 ist die Transformation des Multiroboterfahrbefehls $\dot{\boldsymbol{\xi}}_m$ in den Robotermittelpunkt $\mathbf{p}_{0,i}$ grafisch dargestellt. Der

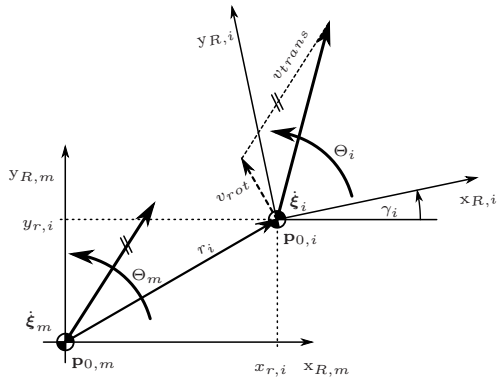


Abbildung 4.6: Transformation des Fahrbefehls $\dot{\boldsymbol{\xi}}_m$

transformierte Fahrbefehl $\dot{\boldsymbol{\xi}}_i$ setzt sich aus einem translatorischen Anteil v_{trans} und einem rotatorischen Anteil v_{rot} zusammen. In Gleichung 4.8 ist die Transformationsvorschrift des Multi-Roboterfahrbefehls $\dot{\boldsymbol{\xi}}_m$ in den Robotermittelpunkt $\mathbf{p}_{0,i}$ angegeben. Der Index i (i=[1,..,n]) wird benötigt, damit bei einem Multiroboterfahrwerk, das aus n Roboterfahrwerken aufgebaut ist, die einzelnen Roboterfahrwerke i unterschieden werden können.

$$\dot{\boldsymbol{\xi}}_i = \mathbf{T}_{\gamma,i} \left(\dot{\boldsymbol{\xi}}_m + \underbrace{\dot{\Theta}_m \cdot \begin{bmatrix} -y_{r,i} \\ x_{r,i} \\ 0 \end{bmatrix}}_{v_{rot}} \right) \tag{4.8}$$

Die Matrix $\mathbf{T}_{\gamma,i}$ (4.9) wird auch als homogene Rotationsmartix bezeichnet.

$$\mathbf{T}_{\gamma,i} = \begin{bmatrix} \cos(\gamma_i) & \sin(\gamma_i) & 0 \\ -\sin(\gamma_i) & \cos(\gamma_i) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.9}$$

Vorteil der virtuellen Verschiebung des Robotermittelpunktes $\mathbf{p}_{0,i}$ ist, dass die Modelldefinitionen der Roboterfahrwerke, inklusive Offsetwinkel $\beta_{os(k_i,i)}$ der Radwaben, nicht geändert werden müssen. Eine ausführliche mathematische Herleitung dieser Transformationsvorschrift ist in [14] (Kapitel 6.2) nachzulesen.

4.1.9 Offsetwinkel $\beta_{os(k_i,i)}$ einer Radwabe

Bei jeder Radwabe ist an der Radlenkeinheit eine Lichtschranke befestigt, die es ermöglicht, dass die Radlenkeinheit eine Referenzposition $\beta_{ref.}$ besitzt. Diese Referenzposition $\beta_{ref.}$ befindet sich an der gegenüberliegenden Ecke (siehe Abbildung 4.7) des Radlenkmotors (1). Der Offsetwinkel $\beta_{os(k_i,i)}$ ist nun jener Winkel, der zur Referenzposition $\beta_{ref.}$ hinzu addiert werden muss, damit die Radachse, unabhängig von der Einbaulage der Radwabe, in Richtung Robotermittelpunkt $\mathbf{p}_{0,i}$ zeigt, und das in einer Weise, dass sich das Fahrwerksrad bei positiver Raddrehrichtung (gegen den Uhrzeigersinn) im Uhrzeigersinn um den Robotermittelpunkt $\mathbf{p}_{0,i}$ drehen würde. Der Lenkwinkel der sich ergibt, wenn der Offsetwinkel $\beta_{os(k_i,i)}$ von der Radlenkeinheit angefahren wurde, wird als Lenkwinkel-Nullposition $\beta_{k_i,i}$=0.0° definiert. Die Offsetwinkel $\beta_{os(k_i,i)}$ werden in der Modelldefinition des Roboterfahrwerkes gespeichert und bei der Referenzierung der Radwaben zur Referenzposition $\beta_{ref.}$ hinzuaddiert.

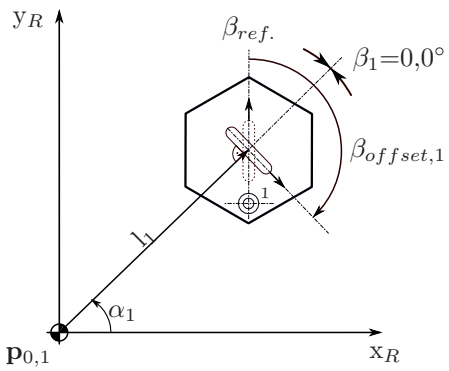


Abbildung 4.7: Definition des Offsestwinkels $\beta_{os(k_i,i)}$

4.1.10 Bewegungsräume

Allgemein dient ein Bewegungsraum dazu, die ausführbaren oder nicht ausführbaren Bewegungen eines Roboterfahrwerkes in der Bewegungsebene mathematisch zu beschreiben. Da eine ebene Bewegung 3 Freiheitsgrade besitzt, die unabhängig voneinander gewählt werden können, entspricht der Bewegungsraum dem $\mathbb{R}^3$. Der Bewegungsraum wird dabei als (3×3) - Matrix dargestellt. Die Spaltenvektoren des Bewegungsraumes entsprechen dabei den Geschwindigkeitsvektoren $\dot{\boldsymbol{\xi}}$ und können als Basisvektoren des $\mathbb{R}^3$ Unterraumes verstanden werden. Besitzt die (3×3) - Matrix des Bewegungsraumes vollen Rang, dann spannen die 3 Basisvektoren einen 3 dimensionalen Raum auf. In diesem Fall können alle drei Parameter eines Fahrbefehls $\dot{\boldsymbol{\xi}}$ frei gewählt werden. Um zu prüfen, ob ein vorliegender Fahrbefehl $\dot{\boldsymbol{\xi}}$ von einem Roboterfahrwerk ausgeführt werden kann, wird der Raum der „Zulässigen und steuerbaren Bewegungen“, auch als Raum $\mathbf{B}$ bezeichnet, benötigt. Kann der auszuführende Fahrbefehl $\dot{\boldsymbol{\xi}}$ als Linearkombination der 3 Basisvektoren (3 Spaltenvektoren von $\mathbf{B}$) dargestellt werden, dann kann der Fahrbefehl $\dot{\boldsymbol{\xi}}$ vom Roboterfahrwerk ausgeführt werden und es gilt $\dot{\boldsymbol{\xi}} \in \mathbf{B}$. Um den Raum der zulässigen und steuerbaren Bewegungen (Raum $\mathbf{B}$) eines Roboterfahrwerkes berechnen zu können, müssen im vorhinein folgende drei Räume bestimmt werden:

$\mathbf{Z}$ ... Raum der zulässigen Bewegungen
$\bar{\mathbf{S}}$ ... Raum der nicht steuerbaren Bewegungen
$\mathbf{S}$ ... Raum der steuerbaren Bewegungen

Über das Robotermodell beziehungsweise die qualitativen Roll- und Gleitbedingungsmatrizen $\mathbf{J}_q$ und $\mathbf{C}_q$ (siehe A.2) des Roboterfahrwerkes, kann über die Gleichungen 4.10 und

4.11 der Raum $\mathbf{Z}$ und $\bar{\mathbf{S}}$ bestimmt werden (siehe [9]).

$$\mathbf{Z} = Kern\left(\mathbf{C}_q\right) \tag{4.10}$$

$$\bar{\mathbf{S}} = Kern\left(\mathbf{J}_q\right) \tag{4.11}$$

$$\mathbf{S} = Kern\left(\bar{\mathbf{S}}^T\right) \tag{4.12}$$

Unter dem „Kern“ einer Matrix wird die Lösungsmenge des zu der Matrix gehörigen homogenen linearen Gleichungssystems verstanden.

$$\mathbf{A} \in \mathbb{R}^{n \times n}, \qquad Kern(\mathbf{A}) = \{v \in \mathbb{R}^n \mid \mathbf{A}\mathbf{v} = 0\} \tag{4.13}$$

Handelt es sich bei der Matrix $\mathbf{A}$ um eine $(3 \times 3]$ -Matrix, so entspricht der Kern der Matrix $\mathbf{A}$ der Lösungsmenge des Gleichungssystems:

$$\mathbf{A}\mathbf{x} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.14}$$

mit

$$Kern(\mathbf{A}) = [\mathbf{x}_1, \ldots \mathbf{x}_n] \tag{4.15}$$

Um aus dem Raum $\mathbf{Z}$ und $\bar{\mathbf{S}}$ den Raum $\mathbf{B}$ berechnen zu können, müssen 2 Fallunterscheidungen getroffen werden:

$$\textbf{Fall A:} \quad \mathbf{Z} \cap \bar{\mathbf{S}} = 0 \qquad \Rightarrow \qquad \mathbf{B} = \mathbf{Z} \tag{4.16}$$

$$\textbf{Fall B:} \quad \mathbf{Z} \cap \bar{\mathbf{S}} \neq 0 \qquad \Rightarrow \qquad \mathbf{B} = \mathbf{Z} \cap \mathbf{S} \tag{4.17}$$

Mit dem berechneten Raum $\mathbf{B}$ kann über die Bedingung 4.18 bestimmt werden, ob der auszuführende Fahrbefehl $\dot{\boldsymbol{\xi}}$ Teil des Raumes $\mathbf{B}$ ist ($\dot{\boldsymbol{\xi}} \in \mathbf{B}$) und somit für das Roboterfahrwerk ausführbar ist.

$$Rang([\mathbf{B}, \dot{\boldsymbol{\xi}}]) = Rang\left(\mathbf{B}\right) \tag{4.18}$$

Wenn nicht, dann besteht unter Umständen die Möglichkeit, dass der Fahrfefehl $\dot{\boldsymbol{\xi}}$ angepasst werden kann. Unter der Rekonfiguration eines Fahrbefehls wird verstanden, dass einzelne Parameter (meist der Winkel Θ_R) des ursprünglichen Fahrbefehls $\dot{\boldsymbol{\xi}}_{orig.}$ so abgeändert werden, dass der rekonfigurierte Fahrbefehls $\dot{\boldsymbol{\xi}}_{rek.}$ vom Roboterfahrwerk ausgeführt wer-

den kann. Die Ausführung eines rekonfigurierten Fahrbefehls, bei dem zum Beispiel die Winkelgeschwindigkeit $\dot{\Theta}_R$ verändert wurde, ist mit der Einschränkung verbunden, dass das Roboterfahrwerk die vorgegebene Sollwinkelgeschwindigkeit $\dot{\Theta}_{R,orig}$ nicht mehr halten kann und somit die Ausrichtung des Roboterfahrwerkes zum globalen Roboterkoordinatensystem Σ_I geändert wird. Eine ausführliche Beschreibung zum Thema „Rekonfiguration von Fahrbefehlen“ wird in [14] (Kapitel 5.4.3) gegeben. Im Multiroboterfall müssen bei der Berechnung des Raumes $\mathbf{B}$ noch zusätzliche Faktoren berücksichtigt werden. Nachdem die Räume $\mathbf{Z}_i$ und $\bar{\mathbf{S}}_i$ (i=[1,..,n]) für alle n Roboterfahrwerke berechnet wurden, müssen die Bewegungsräume der einzelnen Roboterfahrwerke i, in den Multiroboter-Mittelpunkt $\mathbf{p}_{0,m}$ transformiert werden. Bei der Transformation von Bewegungsräumen müssen die Transformationsparameter aus Sicht des lokalen Roboterkoordinatensystems $\Sigma_{R,i}$ bestimmt werden. Die Transformation eines Bewegungsraumes erfolgt schrittweise, indem jeder einzelne Spaltenvektor der (3×3) - Matrix des Bewegungsraumes über die Transformationsforschrift 4.8 in den neuen Multirobotermittelpunkt $\mathbf{p}_{0,m}$ transformiert wird. In Gleichung 4.19 ist beispielhaft der Bewegungsraum $\mathbf{Z}$ mit seinen 3 Geschwindigkeitsvektoren $\dot{\boldsymbol{\xi}}$, die auch als Basisvektoren des Raumes $\mathbf{Z}$ interpretiert werden können, dargestellt.

$$\mathbf{Z} = \left[\dot{\boldsymbol{\xi}}_{Z,1}, \dot{\boldsymbol{\xi}}_{Z,2}, \dot{\boldsymbol{\xi}}_{Z,3}\right] \tag{4.19}$$

Nachdem die Bewegungsräume $\mathbf{Z}_i$ und $\bar{\mathbf{S}}_i$ in den Multiroboter-Mittelpunkt $\mathbf{p}_{0,m}$ transformiert wurden,

$$\begin{aligned} \mathbf{Z}_i &\Rightarrow \mathbf{Z}_i^* \\ \bar{\mathbf{S}}_i &\Rightarrow \bar{\mathbf{S}}_i^* \end{aligned}$$

müssen die transformierten Bewegungsräume $\mathbf{Z}_i^*$ und $\bar{\mathbf{S}}_i^*$ aller n Roboterfahrwerke zu den Multiroboter-Bewegungsräumen $\mathbf{Z}_m$ und $\bar{\mathbf{S}}_m$ geschnitten werden (siehe Gleichung 4.20 und 4.21).

$$\mathbf{Z}_m = \mathbf{Z}_1^* \cap \mathbf{Z}_2^* \cap ... \cap \mathbf{Z}_n^* \tag{4.20}$$

$$\bar{\mathbf{S}}_m = \bar{\mathbf{S}}_1^* \cap \bar{\mathbf{S}}_2^* \cap ... \cap \bar{\mathbf{S}}_n^* \tag{4.21}$$

Für die Berechnung der Schnittbasis zweier Teilräume wird der Zassenhaus-Algorithmus verwendet. Auf die Funktion dieses Algorithmus wird hier nicht näher eingegangen, es wird auf [16] verwiesen. Nachdem der Raum der zulässigen Bewegungen $\mathbf{Z}_m$ und der Raum der nicht steuerbaren Bewegungen $\bar{\mathbf{S}}_m$ für das Multiroboterfahrwerk bestimmt wurde, kann über die Fallentscheidung 4.16 und 4.17 der Raum der zulässigen und steuerbaren

Bewegungen $\mathbf{B}_m$, für das Multiroboterfahrwerk berechnet werden. Anhand des Raumes $\mathbf{B}_m$ kann über die Bedingung 4.18 beurteilt werden, ob ein vorliegender Multiroboter-Fahrbefehl $\dot{\boldsymbol{\xi}}_m$ vom Multiroboterfahrwerk ausgeführt werden kann.

4.1.11 Multiroboterkonfiguration „Vollständig verteilter Fall“

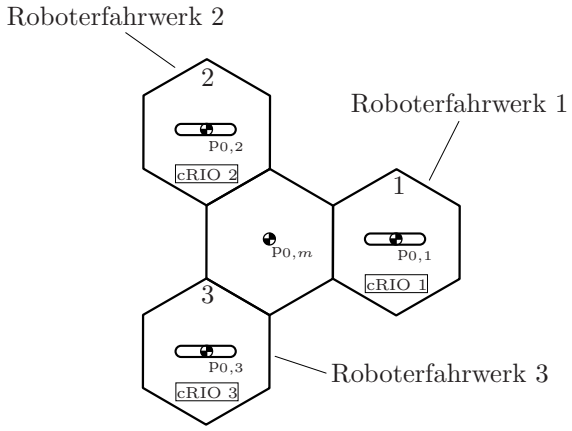


Abbildung 4.8: Multiroboterkonfiguration „Vollständig verteilter Fall“

Die Multiroboterkonfiguration „Vollständig verteilter Fall“ ist ein Spezialfall eines Multiroboter-Szenarios, bei dem alle beteiligten Roboterfahrwerke nur aus einer Radwabe mit Recheneinheit (cRIO) bestehen. Wie aus der Bezeichnung „Vollständig verteilter Fall“ abzuleiten ist, stellt diese Multiroboterkonfiguration den maximal verteilten Fall der Recheneinheiten dar. Das heißt, eine weitere Verteilung der Recheneinheiten ist nicht mehr möglich. In Abbildung 4.8 ist eine Multiroboterkonfiguration im „Vollständig verteilter Fall“ dargestellt. Wenn der Robotermittelpunkt $\mathbf{p}_{0,i}$ im Zentrum (exakt auf der Raddrehachse) einer Radwabe liegt, was im Falle der Multiroboterkonfiguration „Vollständig verteilter Fall“ aus symmetrischen Gründen nahe liegend ist, dann sind die Abstandsparameter l_i und α_i, welche die Lage des Roboterrades zum lokalen Roboterkoordinatensystem $\Sigma_{R,i}$ festlegen, unabhängig von der Lage des $\Sigma_{R,i}$-Koordinatensystems, immer null . Das hat zur Folge, dass die Lage des Roboterrades bezogen auf das lokalen Roboterkoordinatensystem $\Sigma_{R,i}$ nicht eindeutig festgelegt werden kann. Um dieses Problem zu lösen, müssen zwei Vorgaben getroffen werden:

1. Die Lage des lokalen Roboterkoordinatensystems $\Sigma_{R,i}$ muss so gewählt werden, dass sich der Ursprung des $\Sigma_{R,i}$-Koordinatensystems im Robotermittelpunkt $\mathbf{p}_{0,i}$ befindet und die x-Achse des $\Sigma_{R,i}$-Koordinatensystems durch die Referenzposition $\beta_{ref.}$ der Radwabe geht (siehe 4.1.9).

2. Der Offsetwinkel $\beta_{os(k_i,i)}$ beträgt für jede Radwabe $-\pi/2$ rad (-1.57 rad) bzw. -90°.

Durch die Referenzposition der Lenkeinheit (Position der Lichtschranke) ist sichergestellt, dass das Fahrwerksrad eine definierte Lenkwinkel-Referenzposition β_{ref} besitzt. Durch die Vorgabe 1 wird erreicht, dass die x-Achse des lokalen $\Sigma_{R,i}$-Roboterkoordinatensystems immer durch diese Referenzposition β_{ref} geht. Dadurch ist die Lage des lokale $\Sigma_{R,i}$-Roboterkoordinatensystems auch ohne Radabstandsparameter l_i und α_i eindeutig festgelegt. Mit der Vorgabe 2 wird die Offsetwinkeldefinition aus (4.1.9) erfüllt. In Abbildung

(a) Robotermittelpunkt $\mathbf{p}_{0,i}$ außerhalb des Raddrehachse

(b) Robotermittelpunkt $\mathbf{p}_{0,i}$ im Radwabenzentrum

Abbildung 4.9: Offsetwinkel $\beta_{os(k_i,i)}$ - Radrehachse im Robotermittelpunkt $\mathbf{p}_{0,i}$

4.9a ist die Raddrehachse ein infinitesimal kleines Stück dl außerhalb des Robotermittelpunktes $\mathbf{p}_{0,i}$ (l_i=dl, α_i=0). Die Referenzposition β_{ref} des Rades zeigt ebenfalls in die Richtung der $x_{R,i}$-Achse. In diesem Fall muss der Offsetwinkel $\beta_{os(k_i,i)}$ mit $-\pi/2$ rad (-1.57 rad) bzw. -90°. gewählt werden, um die Offsetwinkeldefinition (siehe Punkt 4.1.9) zu erfüllen. Wird der infinitesimal kleine Abstand l_i zu null, dann behält der ermittelte Offsetwinkel $\beta_{os(k_i,i)}$ mit -1.57 rad seine Gültigkeit. In Abbildung 4.9b ist die Lage des lokalen $\Sigma_{R,i}$-Roboterkoordinatensystems und der Offsetwinkel $\beta_{os(k_i,i)}$ für den Fall dargestellt, das sich der Robotermittelpunkt $\mathbf{p}_{0,i}$ genau auf der Raddrehachse befindet.

4.1.12 Inverskinematik

Die Inverskinematik stellt das mathematische Werkzeug zur Verfügung, um aus einem Fahrbefehl $\dot{\boldsymbol{\xi}}_i$ die Radlenkwinkel $\beta_{k_i,i}$ und die Raddrehzahlen $\omega_{k_i,i}$ der Roboterfahrwerksräder zu berechnen. Die Berechnung der Radlenkwinkel $\beta_{k_i,i}$ und der Raddrehzahlen $\omega_{k_i,i}$ erfolgt über den Momentanpol $\mathbf{p}_{MP,i}$. Dabei muss unterschieden

werden, ob der Momentanpol $\mathbf{p}_{MP,i}$ im Endlichen oder im Unendlichen liegt. Auf eine genaue Herleitung der Inverskinematik wird in dieser Arbeit verzichtet, es wir auf [14] (Kapitel 5.4.5) verwiesen.

4.2 Rechenbeispiel - Multiroboterfahrwerk

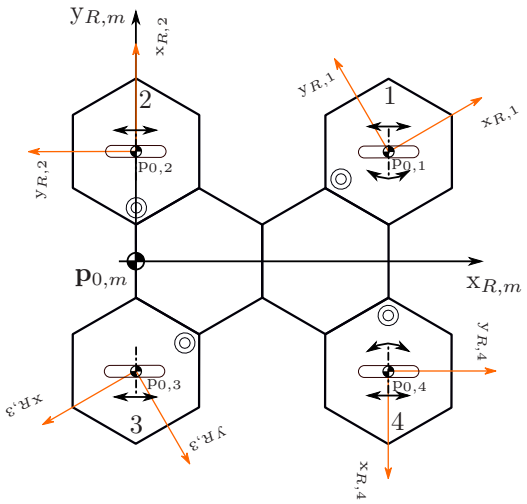


Abbildung 4.10: Multiroboterfahrwerk aus 4 eigenständigen Radwaben

Am Beispiel der Multiroboterkonfiguration „Vollständig verteilter Fall“ von Abbildung 4.10 sollen die notwendigen Rechenschritte, von der Bestimmung der Offsetwinkel $\beta_{os(k_i,i)}$ bis zur Berechnung der Radlenkwinkel $\beta_{k_i,i}$ durch Vorgabe eines Fahrbefehls $\dot{\boldsymbol{\xi}}_m$, anhand eines Zahlenbeispieles verdeutlicht werden. In Abbildung 4.10 ist ein Multi-Roboterfahrwerk mit vier (einrädrigen) Roboterfahrwerken dargestellt. Die eingestellten Radlenkwinkel $\beta_{1,1}$ und $\beta_{1,2}$ von Roboterfahrwerk 2 und 3 sind fix vorgegeben und ändern sich auch während der Fahrt des Multiroboterfahrwerkes nicht. Die Lage des lokalen Multiroboter-Mittelpunktes $\mathbf{p}_{0,m}$ und des lokalen Multiroboter-Koordinatensystems $\Sigma_{R,m}$ wurden frei gewählt. Als erstes werden für das vorliegende Multi-Roboterfahrwerk die Offsetwinkel $\beta_{os(k_i,i)}$ der Roboterfahrwerke bestimmt. Bei der Multiroboterkonfiguration „Vollständig verteilter Fall“ müssen für die Ermittlung der Offsetwinkel $\beta_{os(k_i,i)}$ die Vorgaben aus Punkt 4.1.11 erfüllt werden, damit später über die Inverskinematik die richtigen Radlenkwinkel $\beta_{k_i,i}$ berechnet werden können. Das heißt, der Offsetwinkel $\beta_{os(k_i,i)}$ beträgt für jedes (einrädrige) Roboterfahrwerk -1.57 rad bzw. -90°. In Abbildung 4.11 sind die Offsetwinkel $\beta_{os(k_i,i)}$ aller vier Roboterfahrwerke nochmals grafisch dargestellt. Nachdem die Offsetwinkel $\beta_{os(k_i,i)}$ festgelegt wurden, kann die Modelldefinition für jedes Roboterfahrwerk be-

Fahrwerk 1:	$\beta_{os(1,1)}$ = -1.57 rad
Fahrwerk 2:	$\beta_{os(1,2)}$ = -1.57 rad
Fahrwerk 3:	$\beta_{os(1,3)}$ = -1.57 rad
Fahrwerk 4:	$\beta_{os(1,4)}$ = -1.57 rad

Tabelle 4.1: Offsetwinkel $\beta_{os(k_i,i)}$ der Roboterfahrwerke

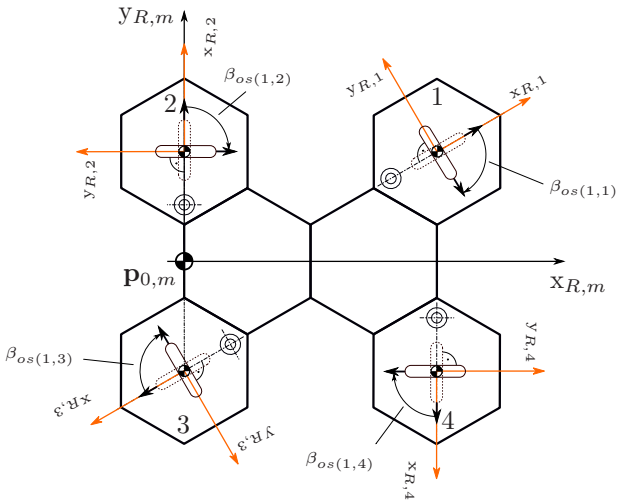


Abbildung 4.11: Festlegung der Offsetwinkel $\beta_{os(k_i,i)}$

stimmt werden. Im Falle der Multiroboterkonfiguration „Vollständig verteilter Fall“ ist das relativ einfach, da nur die Radtypen und die Radoffsetwinkel $\beta_{os(k_i,i)}$ eingetragen werden müssen. Bei den Radwaben mit fest eingestellten Radlenkwinkel, den so genannten „Fixed Wheels“, müssen noch zusätzlich die fixen Radlenkwinkel $\beta_{k_i,i}$ in die Modelldefinition eingetragen werden. In der Tabelle 4.2 sind die Daten angeführt, die in der Modelldefinition der Roboterfahrwerke gespeichert werden. Zusätzliche Daten wie Raddurchmesser und ID-Adressen der Motorregler werden hier nicht angeführt. Über die festgelegten Modelldefinitionen der Roboterfahrwerke können die Robotermodelle bzw. die qualitativen Roll- und Gleitbedingungsmatrizen $\mathbf{J}_{q,i}$ und $\mathbf{C}_{q,i}$ der Roboterfahrwerke über die Tabelle A.2 bestimmt werden. In der Tabelle 4.3 sind die ermittelten Matrizen aufgrund der Modelldefinitionen der Roboterfahrwerke angegeben. Mittels der bestimmten qualitativen Roll- und Gleitbedingungsmatrizen $\mathbf{J}_{q,i}$ und $\mathbf{C}_{q,i}$ können über die Gleichungen 4.10 und 4.11 die Räume $\mathbf{Z}_i$ und $\bar{\mathbf{S}}_i$ der Roboterfahrwerke berechnet werden. In Tabelle 4.4 sind die Bewegungsräume für die Roboterfahrwerke dargestellt. Bevor die Bewegungsräume $\mathbf{Z}_i$ und $\bar{\mathbf{S}}_i$ der einzelnen Roboterfahrwerke, die sich auf den lokalen Robotermittelpunkt $\mathbf{p}_{0,i}$ beziehen, in den Multiroboter-Mittelpunkt $\mathbf{p}_{0,m}$ transformiert werden können, müssen noch die

Fahrwerk 1: „Steered-Wheel“ l_1=0 m α_1=0 rad $\beta_{os(1,1)}$=-1.57 rad	**Fahrwerk 2:** „Fixed-Wheel“ l_2=0 m α_2=0 rad $\beta_{os(1,2)}$=-1.57 rad $\beta_{1,2}$=0 rad
Fahrwerk 3: „Fixed-Wheel“ l_3=0 m α_3=0 rad $\beta_{os(1,3)}$=-1.57 rad $\beta_{1,3}$=-2.09 rad	**Fahrwerk 4:** „Steered-Wheel“ l_4=0 m α_4=0 rad $\beta_{os(1,4)}$=-1.57 rad

Tabelle 4.2: Modelldefinitionen der Roboterfahrwerke

Fahrwerk 1: $\mathbf{J}_{q,1}=\begin{bmatrix}0 & -1 & 0\\ 1 & 0 & 0\end{bmatrix}$ $\mathbf{C}_{q,1}=\begin{bmatrix}0 & 0 & 0\end{bmatrix}$	**Fahrwerk 2:** $\mathbf{J}_{q,2}=\begin{bmatrix}0 & -1 & 0\end{bmatrix}$ $\mathbf{C}_{q,2}=\begin{bmatrix}1 & 0 & 0\end{bmatrix}$
Fahrwerk 3: $\mathbf{J}_{q,3}=\begin{bmatrix}-0.866 & 0.5 & 0\end{bmatrix}$ $\mathbf{C}_{q,3}=\begin{bmatrix}-0.5 & -0.866 & 0\end{bmatrix}$	**Fahrwerk 4:** $\mathbf{J}_{q,4}=\begin{bmatrix}0 & -1 & 0\\ 1 & 0 & 0\end{bmatrix}$ $\mathbf{C}_{q,4}=\begin{bmatrix}0 & 0 & 0\end{bmatrix}$

Tabelle 4.3: Qualitative Roll- und Gleitbedingungsmatrizen der Roboterfahrwerke

Transformationsparameter l_i, α_i und γ_i bestimmt werden. Die Parameter l_i und α_i legen die Position des Multiroboter-Mittelpunkt $\mathbf{p}_{0,m}$ bezogen auf das lokale Roboterkoordinatensystem $\Sigma_{R,i}$ fest und der Parameter γ_i legt eine Verdrehung des lokale Multiroboter-Koordinatensystem $\Sigma_{R,m}$ gegenüber dem lokale Roboterkoordinatensystems $\Sigma_{R,i}$ fest. In Abbildung 6.2.2 sind die Transformationsparameter beispielhaft für das Roboterfahrwerk 1 grafisch dargestellt. In Tabelle 4.5 sind die ermittelten Transformationsparameter für die 4 Roboterfahrwerke angegeben. Mittels den ermittelten Transformationsparameter und der Transformationsvorschrift von Gleichung 4.8 können die Bewegungsräume $\mathbf{Z}_i$ und $\bar{\mathbf{S}}_i$

Fahrwerk 1: $\mathbf{Z}_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ $\bar{\mathbf{S}}_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$	**Fahrwerk 2:** $\mathbf{Z}_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$ $\bar{\mathbf{S}}_2 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$
Fahrwerk 3: $\mathbf{Z}_3 = \begin{bmatrix} 0 & -0.866 & 0 \\ 0 & 0.5 & 0 \\ 1 & 0 & 0 \end{bmatrix}$ $\bar{\mathbf{S}}_3 = \begin{bmatrix} 0 & 0.5 & 0 \\ 0 & 0.866 & 0 \\ 1 & 0 & 0 \end{bmatrix}$	**Fahrwerk 4:** $\mathbf{Z}_4 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ $\bar{\mathbf{S}}_4 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$

Tabelle 4.4: Bewegungsraum $\mathbf{Z}_i$ und $\bar{\mathbf{S}}_i$ der Roboterfahrwerke

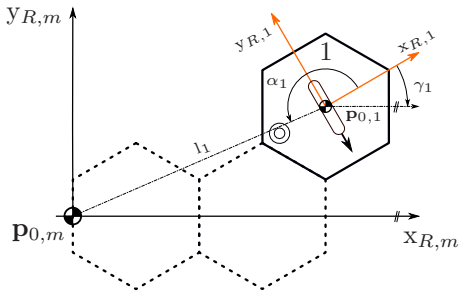


Abbildung 4.12: Festlegung der Transformationsparameter für die Bewegungsräume

in den Multiroboter-Mittelpunkt $\mathbf{p}_{0,m}$ transformiert werden. Die transformierten Bewegungsräume $\mathbf{Z}_i{}^*$ und $\bar{\mathbf{S}}_i{}^*$ sind in der Tabelle 4.6 angegeben. Für die Bestimmung der Multiroboter-Bewegungsräume $\mathbf{Z}_m$ und $\bar{\mathbf{S}}_m$ müssen die transformierten Bewegungsräume $\mathbf{Z}_i^*$ und $\bar{\mathbf{S}}_i^*$ der Roboterfahrwerke miteinander geschnitten werden (siehe Gleichung 4.22 und 4.23). Für die Berechnung der Schnittbasis von zwei Bewegungsräumen kann der Zassenhaus-Algorithmus verwendet werden. Auf diesen Algorithmus soll hier nicht näher eingegangen werden, es wird auf [16] verwiesen.

$$\mathbf{Z}_m = \mathbf{Z}_1^* \cap \mathbf{Z}_2^* \cap \mathbf{Z}_3^* \cap \mathbf{Z}_4^* \tag{4.22}$$

Fahrwerk 1: l_1=0.567 m α_1=3.027 rad γ_1=-0.524 rad	**Fahrwerk 2:** l_2=0.225 m α_2=3.141 rad γ_2=-1.570 rad
Fahrwerk 3: l_3=0.225 m α_3=-2.094 rad γ_3=2.618 rad	**Fahrwerk 4:** l_4=0.567 m α_4=-1.979 rad γ_4=1.570 rad

Tabelle 4.5: Transformationsparameter für die Bewegungsräume $\mathbf{Z}_i$ und $\bar{\mathbf{S}}_i$

Fahrwerk 1: $\mathbf{Z}_1^*=\begin{bmatrix}0.866 & -0.5 & 0.225\\ 0.5 & 0.866 & -0.520\\ 0 & 0 & 1\end{bmatrix}$ $\bar{\mathbf{S}}_1^*=\begin{bmatrix}0.225 & 0 & 0\\ 0.52 & 0 & 0\\ 1 & 0 & 0\end{bmatrix}$	**Fahrwerk 2:** $\mathbf{Z}_2^*=\begin{bmatrix}0.225 & -1 & 0\\ 0 & 0 & 0\\ 1 & 0 & 0\end{bmatrix}$ $\bar{\mathbf{S}}_2^*=\begin{bmatrix}0.225 & 0 & 0\\ 0 & 1 & 0\\ 1 & 0 & 0\end{bmatrix}$
Fahrwerk 3: $\mathbf{Z}_3^*=\begin{bmatrix}-0.225 & 1 & 0\\ 0 & 0 & 0\\ 1 & 0 & 0\end{bmatrix}$ $\bar{\mathbf{S}}_3^*=\begin{bmatrix}-0.225 & 0 & 0\\ 0 & -1 & 0\\ 1 & 0 & 0\end{bmatrix}$	**Fahrwerk 4:** $\mathbf{Z}_4^*=\begin{bmatrix}0 & 1 & -0.225\\ -1 & 0 & -0.52\\ 0 & 0 & 1\end{bmatrix}$ $\bar{\mathbf{S}}_4^*=\begin{bmatrix}-0.225 & 0 & 0\\ -0.52 & 0 & 0\\ 1 & 0 & 0\end{bmatrix}$

Tabelle 4.6: Transformierten Bewegungsräume $\mathbf{Z}_i^*$ und $\bar{\mathbf{S}}_i^*$

$$\bar{\mathbf{S}}_m = \bar{\mathbf{S}}_1^* \cap \bar{\mathbf{S}}_2^* \cap \bar{\mathbf{S}}_3^* \cap \bar{\mathbf{S}}_4^* \tag{4.23}$$

Die ermittelten Bewegungsräume $\mathbf{Z}_m$ und $\bar{\mathbf{S}}_m$ für das Multiroboterfahrwerk sind in Gleichung 4.24 und 4.25 angegeben.

$$\mathbf{Z}_m = \begin{bmatrix}1 & 0 & 0\\ 0 & 0 & 0\\ 0 & 1 & 0\end{bmatrix} \tag{4.24}$$

$$\bar{\mathbf{S}}_m = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{4.25}$$

Anhand der Multiroboter-Bewegungsräume $\mathbf{Z}_m$ und $\bar{\mathbf{S}}_m$ kann nun über die Fallentscheidung 4.16 und 4.17 der Bewegungsraum $\mathbf{B}_m$, der Raum der zulässigen und steuerbaren Bewegungen, für das Multiroboterfahrwerk berechnet werden. Werden die Spaltenvektoren des ermittelten Bewegungsraumes $\mathbf{B}_m$ (Gleichung 4.26) näher betrachtet, dann ist zu erkennen, dass sich das Multiroboterfahrwerk uneingeschränkt in Richtung der $\mathrm{x}_{R,m}$-Achse bewegen oder um den Multirobotermittelpunktes $\mathbf{p}_{0,m}$ drehen kann.

$$\mathbf{B}_m = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{4.26}$$

Dach dem der Raum für die zulässigen und steuerbaren Bewegungen $\mathbf{B}_m$ für das Multiroboterfahrwerk ermittelt wurde, kann nun ein vorliegender Multiroboter-Fahrbefehl $\dot{\boldsymbol{\xi}}_m$ auf seine Ausführbarkeit hin überprüft werden. Für dieses Rechenbeispiel soll ein Fahrbefehl vorgegeben werden, bei dem sich das Multiroboterfahrwerk um den Multiroboter-Mittelpunkt $\mathbf{p}_{0,m}$, gegen den Uhrzeigersinn, dreht. Der erforderliche Fahrbefehl $\dot{\boldsymbol{\xi}}_m$ dafür, ist in 4.27 angegeben.

$$\dot{\boldsymbol{\xi}}_m = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{4.27}$$

Nun muss überprüft werden, ob das Multiroboterfahrwerk den vorliegenden Fahrbefehl $\dot{\boldsymbol{\xi}}_m$ (4.27) ausführen kann. Hierfür muss gelten, dass der Fahrbefehl $\dot{\boldsymbol{\xi}}_m \in \mathbf{B}_m$ ist, oder anders ausgedrückt, der Fahrbefehl $\dot{\boldsymbol{\xi}}_m$ muss als Linearkombination von einem der 3 Spalten- oder Zeilenvektoren des Bewegungsraumes $\mathbf{B}_m$ darstellbar sein. Ob der Fahrbefehl $\dot{\boldsymbol{\xi}}_m$ nun Teil des Bewegungsraumes $\mathbf{B}_m$ ist, kann mittels Gleichung 4.28 überprüft werden. In diesem Fall ist leicht zu erkennen, dass der vorliegende Fahrbefehl $\dot{\boldsymbol{\xi}}_m$ aus Gleichung 4.27 dem zweiten Spaltenvektor des Bewegungsraumes $\mathbf{B}_m$ entspricht. Daher ist der vorliegende Fahrbefehl $\dot{\boldsymbol{\xi}}_m$ Teil des Bewegungsraumes $\mathbf{B}_m$ und kann vom Multiroboterfahrwerk

ausgeführt werden.

$$Rang\left(\begin{bmatrix}1&0&0\\0&0&0\\0&1&0\end{bmatrix},\begin{bmatrix}0\\0\\1\end{bmatrix}\right) = Rang\left(\begin{bmatrix}1&0&0\\0&0&0\\0&1&0\end{bmatrix}\right) \quad \Rightarrow \quad 2=2 \tag{4.28}$$

Jetzt kann der auf seine Ausführbarkeit hin überprüfte Multiroboter-Fahrbefehl $\dot{\boldsymbol{\xi}}_m$, der sich auf den Multiroboter-Mittelpunkt $\mathbf{p}_{0,m}$ bezieht, in die einzelnen Roboterfahrwerks-Mittelpunkte $\mathbf{p}_{0,i}$ transformiert werden. Dazu wird die Transformationsvorschrift von 4.8 verwendet. Bevor der Multiroboter-Fahrbefehl $\dot{\boldsymbol{\xi}}_m$ transformiert werden kann, müssen noch die Transformationsparameter bestimmt werden. Bei der Transformation von Multiroboter-Fahrbefehlen $\dot{\boldsymbol{\xi}}_m$ muss beachtet werden, dass die Transformationsparameter aus Sicht des lokalen Multiroboter-Koordinatensystems $\Sigma_{R,m}$ bestimmt werden müssen. In Abbildung 4.13 sind die Transformationsparameter für den Multiroboter-Fahrbefehl $\dot{\boldsymbol{\xi}}_m$ grafisch für das Roboterfahrwerk 1 dargestellt. Da die Transformationsparameter für

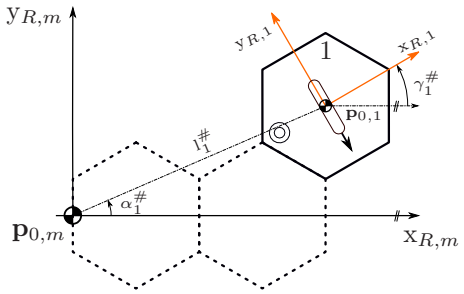


Abbildung 4.13: Transformationsparameter für den Multiroboterfahrbefehl $\dot{\boldsymbol{\xi}}_m$

die Bewegungsräume schon bestimmt wurden (siehe Tabelle 4.5), können diese einfach mittels der Umrechnungsvorschrift von 4.29 invertiert werden, um die notwendigen Transformationsparameter für den Multiroboter-Fahrbefehl $\dot{\boldsymbol{\xi}}_m$ zu erhalten.

$$l_i^{\#} = l_i \qquad \alpha_i^{\#} = \alpha_i \pm \pi \qquad \gamma_i^{\#} = -\gamma_i \tag{4.29}$$

In Tabelle 4.7 sind die Transformationsparameter für den Multiroboter-Fahrbefehl $\dot{\boldsymbol{\xi}}_m$ in die Roboterfahrwerks-Mittelpunkte $\mathbf{p}_{0,i}$ angegeben. Über die Transformationsvorschrift (4.8) kann nun mittels der bestimmten Tranformationsparameter 4.7 der Multiroboter-Fahrbefehl $\dot{\boldsymbol{\xi}}_m$ in die einzelnen Robotermittelpunkte $\mathbf{p}_{0,i}$ transformiert werden. Die transformierten Fahrbefehle für die Roboterfahrwerke sind in Tabelle 4.8 angegebnen. Abschließend sollen aufgrund des vorliegenden Fahrbefehls $\dot{\boldsymbol{\xi}}_m$ (4.27) noch

Fahrwerk 1: $l_1^{\#}$=0.567 m $\alpha_1^{\#}$=0.4085 rad $\gamma_1^{\#}$=0.524 rad	**Fahrwerk 2:** $l_2^{\#}$=0.225 m $\alpha_2^{\#}$=1.570 rad $\gamma_2^{\#}$=1.570 rad
Fahrwerk 3: $l_3^{\#}$=0.225 m $\alpha_3^{\#}$=-1.570 rad $\gamma_3^{\#}$=-2.618 rad	**Fahrwerk 4:** $l_4^{\#}$=0.567 m $\alpha_4^{\#}$=-0.4085 rad $\gamma_4^{\#}$=-1.570 rad

Tabelle 4.7: Transformationsparameter für den Multiroboterfahrbefehl $\dot{\boldsymbol{\xi}}_m$

Fahrwerk 1: $\dot{\boldsymbol{\xi}}_1 = \begin{bmatrix} 0.065 \\ 0.563 \\ 1 \end{bmatrix}$	**Fahrwerk 2:** $\dot{\boldsymbol{\xi}}_2 = \begin{bmatrix} 0 \\ 0.225 \\ 1 \end{bmatrix}$
Fahrwerk 3: $\dot{\boldsymbol{\xi}}_3 = \begin{bmatrix} -0.195 \\ 0.112 \\ 1 \end{bmatrix}$	**Fahrwerk 4:** $\dot{\boldsymbol{\xi}}_4 = \begin{bmatrix} -0.520 \\ 0.225 \\ 1 \end{bmatrix}$

Tabelle 4.8: Transformierter Multiroboterfahrbefehl $\dot{\boldsymbol{\xi}}_m$

die Radlenkwinkel $\beta_{k_i,i}$ der vier Roboterfahrwerke bestimmt werden. Die Berechnung der Radlenkwinkel $\beta_{k_i,i}$ erfolgt über die Inverskinematik (4.1.12). Dabei wird der Momentanpol $\mathbf{p}_{MP,i}$ des Roboterfahrwerkes benötigt. Wenn der Momentanpol $\mathbf{p}_{MP,i}$ im Endlichen liegt, was bei den 4 transformierten Fahrbefehlen $\dot{\boldsymbol{\xi}}_1, .., \dot{\boldsymbol{\xi}}_4$ (siehe Tabelle 4.8) der Fall ist, da $\dot{\Theta}_i \neq 0$ ist, dann kann zu Berechnung der Momentanpole $\mathbf{p}_{MP,i}$ die Gleichung 4.6 verwendet werden. Die Momentanpole $\mathbf{p}_{MP,i}$ der vier Roboterfahrwerke sind in Tabelle 4.9 angegeben. Nachdem die Lage des Momentanpols $\mathbf{p}_{MP,i}$ für jedes Roboterfahrwerk bestimmt wurde, können die Radlenkwinkel $\beta_{k_i,i}$ für die einrädrigen Roboterfahrwerke über die Gleichung 4.30 bestimmt werden. Diese Gleichung gilt jedoch nur für Radwaben, bei denen der Robotermittelpunkt $\mathbf{p}_{0,i}$ genau auf der Raddrehachse liegt.

$$\beta_{k_i,i} = \arctan\left(\frac{y_{MP,i}}{x_{MP,i}}\right) + \pi \tag{4.30}$$

Fahrwerk 1: $\mathbf{p}_{MP,1}= \begin{bmatrix} -0.563 \\ 0.065 \\ 1 \end{bmatrix}$	**Fahrwerk 2:** $\mathbf{p}_{MP,2}= \begin{bmatrix} -0.225 \\ 0 \\ 1 \end{bmatrix}$
Fahrwerk 3: $\mathbf{p}_{MP,3}= \begin{bmatrix} -0.112 \\ 0.195 \\ 1 \end{bmatrix}$	**Fahrwerk 4:** $\mathbf{p}_{MP,4}= \begin{bmatrix} -0.225 \\ -0.520 \\ 1 \end{bmatrix}$

Tabelle 4.9: Momentanpole $\mathbf{p}_{MP,i}$ der Roboterfahrwerke

In Tabelle 4.10 sind die ermittelten Radlenkwinkel $\beta_{k_i,i}$ für die vier Roboterfahrwerke angegeben. In Abbildung 4.14 wurden die Radlenkwinkel $\beta_{k_i,i}$ in das Multiroboterfahr-

Fahrwerk 1:	$\beta_{1,1}$ = 3.03 rad (173.38°)
Fahrwerk 2:	$\beta_{1,2}$ = 3.14 rad (180.00°)
Fahrwerk 3:	$\beta_{1,3}$ = -2.09 rad (-120.00°)
Fahrwerk 4:	$\beta_{1,4}$ = -1.98 rad (-113.46°)

Tabelle 4.10: Radlenkwinkel β_i der Roboterfahrwerke

werk eingezeichnet. Es ist zu erkennen, dass alle Radachsen in Richtung Multiroboter-Mittelpunkt $\mathbf{p}_{0,m}$ zeigen, und die Drehrichtung des Multiroboter-Fahrwerkes gegen den Uhrzeigersinn erfolgt. Das entspricht genau dem vorgegeben Fahrbefehl

$$\dot{\boldsymbol{\xi}}_m = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

4.2.1 Fazit - Rechenbeispiel

Mit diesem Rechenbeispiel könnte gezeigt werden, dass bei der Multiroboterkonfiguration „Vollständig Verteilter Fall“ die Berechnung der Multiroboter-Bewegungsräume, über die herkömmliche Berechnungsvorschrift von Bewegungsräumen (siehe Punkt 4.1.10), durchgeführt werden kann. Auch die Berechnung der Inverskinematik (siehe Punkt 4.1.12) bleibt unverändert. Bei der Multiroboterkonfiguration „Vollständig Verteilter Fall“ muss nur darauf geachtet werden, wenn der Robotermittelpunkt $\mathbf{p}_{0,i}$ exakt auf der Raddrehachse einer

Radwabe liegt. In diesem Fall müssen die Vorgaben, betreffend Lage des lokalen $\Sigma_{R,i}$-Koordinatensystems und Bestimmung des Offsetwinkels $\beta_{os(k_i,i)}$ aus Punkt 4.1.11 erfüllt werden. Ansonst kann die Multiroboterkonfiguration „Vollständig Verteilter Fall“ gleich behandelt werden, wie ein typisches Multiroboter-Szenario, bei dem die Roboterfahrwerke aus mehreren Radwaben bestehen.

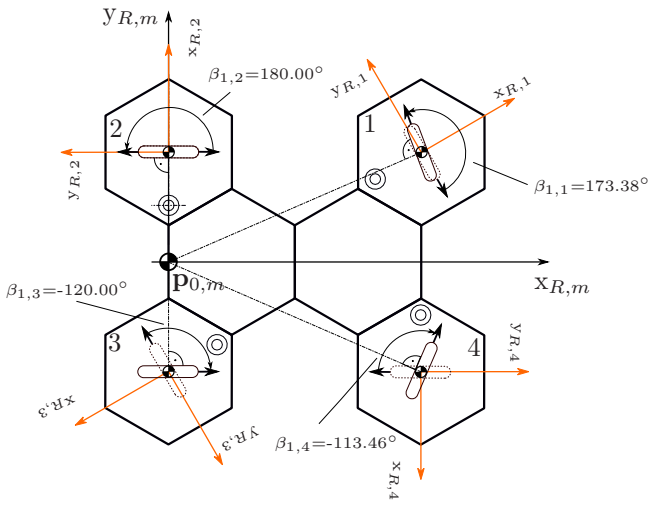


Abbildung 4.14: Ermittelte Radlenkwinkel $\beta_{k_i,i}$ aufgrund des Fahrbefehls $\dot{\boldsymbol{\xi}}_m$

Kapitel 5

Synchronisations-Methoden bei verteilten Systemen

Am Beginn dieses Kapitels soll zunächst geklärt werden, wozu eine Synchronisation zwischen den verteilten Systemen notwendig ist. Bei den verteilten Systemen handelt es sich zum einen um die Recheneinheiten (cRIOs) der Roboterfahrwerke sowie um die Motorsteuergeräte (Whistles) der Radwaben. Unter dem Begriff „Synchronisation“ soll hier verstanden werden, dass sich die Systemuhren der Slave-Teilnehmer (Host-PC und cRIO-Salveroboterfahrwerke) auf die Systemuhr des Master-Teilnehmers (cRIO-Masterroboterfahrwerk) synchronisieren. Eine Synchronisation ist deshalb notwendig, da beim Single- und Multi-Roboterszenario (siehe Kapitel 4) essentielle Funktionen, wie zum Beispiel der Befehl „Fahrt starten“, auf zeitgesteuerten Kommandos basieren. Die Gültigkeit eines zeitgesteuerten Kommandos hängt von einem absoluten Zeitpunkt ab, der dem Kommando zugewiesen wird. Soll nur auf mehreren verteilten Systemen ein zeitgesteuertes Kommando exakt zum selben Zeitpunkt ausgeführt werden, dann ist es essentiell, dass die Systemuhren synchronisiert wurden. Hat keine Synchronisation stattgefunden, dann besteht die Möglichkeit, dass auf einigen Systemen das Kommando gar nicht mehr ausgeführt wird, im speziellen wenn die Gültigkeit des Kommandos auf dem jeweiligen System schon abgelaufen ist, oder das zeitgesteuerte Kommando wird auf dem System zu einem späteren Zeitpunkt, bezogen auf die Master-Systemzeit, ausgeführt. Diese beiden Szenarien sollen durch den Synchronisationsprozess verhindert werden, der üblicherweise in die Initialisierungsphase der verteilten Systeme inkludiert wird.

Der folgende Abschnitte 5.1 beschreibt den Synchronisationsprozess bei einem

Roboterfahrwerk zwischen der Recheneinheit (cRIO) und den Motorregelgeräten (Whistles) der Radwaben. Dieser Synchronisationsprozess ist maßgebend für die zeitgleiche Ausführung der Radlenkwinkel- und der Raddrehzahlvorgaben auf den Radwaben. Der Abschnitt 5.2 beschäftigt sich mit Synchronisations-Methoden zwischen den Recheinheiten (cRIOs) eines Multiroboterfahrwerkes.

5.1 Synchronisation zwischen der Recheneinheit und den Motorregelgeräten

Aufgrund der Forderung, dass zwischen den vorgegebenen Sollwertgrößen interpoliert werden soll, um einen möglichst glatten Bewegungsverlauf zu erzielen, wurde für die Sollwertvorgabe auf den Motorregelgeräten (Whistles) die Position-Time-Table (PPT) verwendet. Die PPT kann als Speicherbereich verstanden werden, der mit einer Sequenz von absoluten Positionswerten beschrieben werden kann, die nach dem Start der PTT, nacheinander zu festen Zeitpunkten angefahren werden. Eine besondere Eigenschaft der PTT ist, dass zwischen den einzelnen Positionsvorgaben durch ein Polynom 3.Ordnung interpoliert wird. Durch diese zusätzlichen Interpolationsstützstellen wird ein sehr glatter Bewegungsablauf für den Lenk- und Antriebsmotor einer Radwabe erzeugt. Eine Beschreibung zur Position-Time-Table ist Abschnitt 2.3.2 zu finden. In Abbildung 5.1 ist der schematische Aufbau eines N-rädrigen Roboterfahrwerkes mit seiner Recheneinheit (cRIO) und den Motorregelgeräten (Whistles) inklusive den integrierten Position-Time-Tables (PTT) dargestellt.

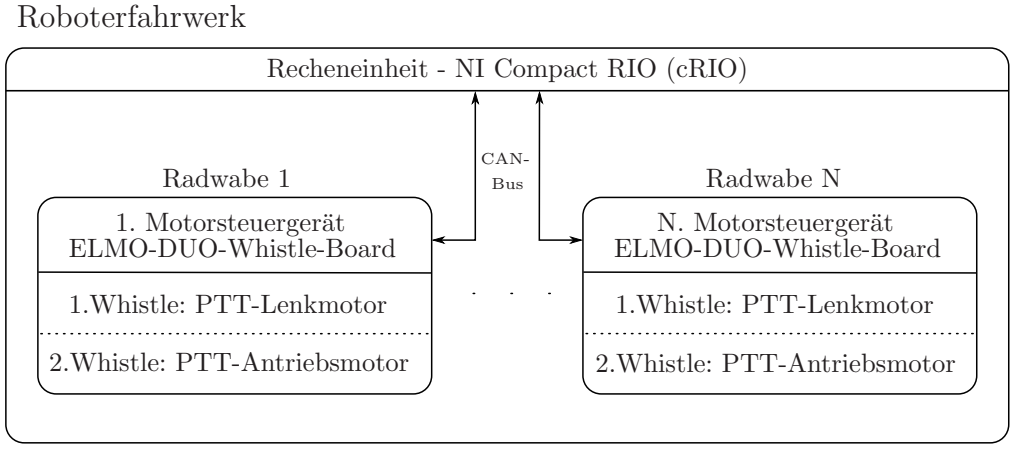


Abbildung 5.1: Beispielkonfiguration Roboterfahrwerk inkl. Whistles

Bei einem Roboterfahrwerk werden periodisch von der Recheneinheit (cRIO) die

Lenkwinkel- und Drehzahlsollwerte für die einzelnen Radwaben mit der Abtastperiode T_d neu berechnet. Durch den synchronen Start der PTTs ist sichergestellt, dass die errechneten Sollwerte, die sich alle auf einen Berechnungszeitpunkt $T_{d,k}$ der Recheneinheit beziehen, auch von den Lenk- und Antriebsmotoren aller Radwaben zum selben Zeitpunkt angefahren werden. Damit die Forderung eines synchronen Starts der PTTs erfüllt wird, kann für den Start der PTTs ein zeitgesteuertes Kommando verwendet werden. Dadurch ist auch der Startzeitpunkt der PTTs von der Übertragungszeit des CAN-Bus-Systems, über das die Recheneinheit (cRIO) mit den Motorregelgeräten (Whistles) verbunden ist, unabhängig. Wie in der Einleitung zu diesem Kapitel schon beschrieben wurde, müssen alle Systemuhren synchronisiert worden sein, damit ein zeitgesteuertes Kommando auf mehreren verteilten Systemen synchron ausgeführt werden kann.

5.1.1 Synchronisationsprinzip - CAN-Bus

Die Motorregelgeräte (Whistles) sind über CAN-Bus mit der Rechenheiteinheit (cRIO) eines Roboterfahrwerkes verbunden. Für die Kommunkation zwischen den ELMO-Whistles und dem cRIO werden die vom Hersteller „ELMO Motion Control GmbH“ definierten „Simple IQ“-Befehle [5] verwendet. Für die Synchronisation der Whistle-Systemuhren an die Systemuhr des cRIOs bietet das CANopen-Protokoll eine Möglichkeit an, bei der mittels des SYNC-Kommandos und eines Timestamps die Zeitdifferenz zwischen der Master(cRIO)- und Slave-Systemuhr(Whistles) ermittelt wird, und anschließend schrittweise korrigiert. In Abbildung 5.2 ist der Synchronisationsvorgang grafisch dargestellt. Immer wenn der Master (Recheneinheit-cRIO) ein SYNC-Kommando sendet, dann speichern die Slaves (Whistles) dessen Empfangszeitpunkt. Innerhalb von 5 Sekunden muss laut „CANopen DS 301 Implementation Guide“ [4] der Master einen Timestamp senden, damit der Synchronisationsprozess startet. Der Timestamp ist ein 32-Bit Wert, der den Wert der Master-Systemuhr in μs, zum Zeitpunkt des zuletzt gesendeten SYNC-Kommandos, enthält. Nach Empfang dieses Timestamps ist der Slave in der Lage, seine Zeitdifferenz bezogen auf den Master zu ermitteln und seine Systemuhr an die des Masters zu synchronisieren. In der Regel muss dieser Vorgang mehrmals wiederholt werden, weil die Systemuhren in den ELMO-Whistles pro SYNC-Timestamp-Paar nur um einige ms geändert werden können. Der CANopen DS 301 Implementation Guide [4] gibt an, dass circa 200 SYNC-Timestamp-Paar gesendet werden müssen, bis sichergestellt ist, dass alle Slaves an den Master synchronisiert wurden. Das ist jedoch nur notwendig, wenn die Sys-

temuhren sehr stark voneinander abweichen. Das kann zum Beispiel der Fall sein, wenn die ELMO-Whistles an verschiedenen Spannungsquellen angeschlossen sind und diese zu sehr unterschiedlichen Zeitpunkten eingeschalten werden. Mit dem SYNC-Timstamp-Verfahren ist beim CAN-Bus eine Synchronität im Bereich von wenigen 10 μs erreichbar bei einer Nachsynchronisation von einigen Sekunden. Die notwendigen SYNC-Timestamp-Paare für die Nachsynchronisation werden dann in die laufende CAN-Bus Kommunikation integriert.

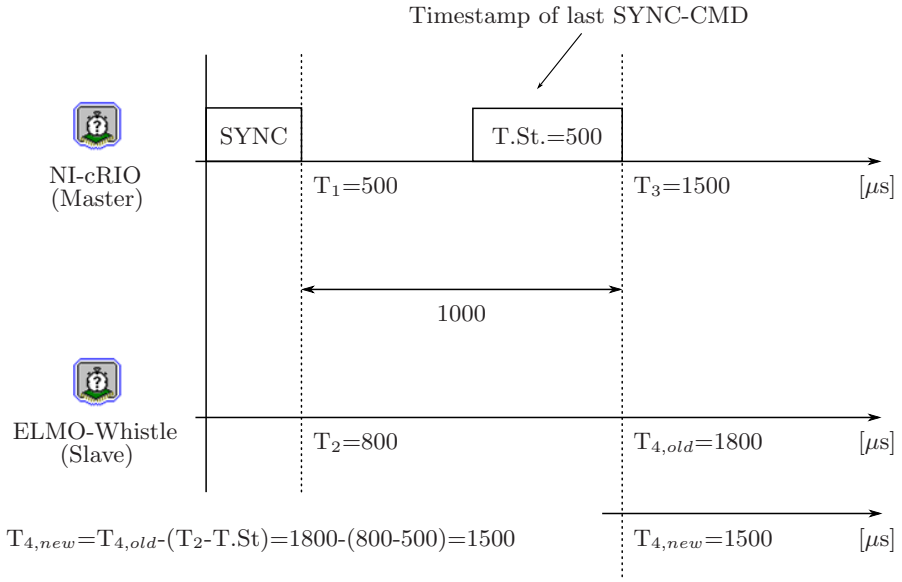


Abbildung 5.2: Synchronisationsvorgang der Slave-Systemuhren an die Master-Systemuhr

Nach der Synchronisation kann für den synchronen Start der PTTs das zeitgesteuerte Simple-IQ-Kommando „Begin Motion at defined Time“ [5] (BT = absoluter Startzeitpunkt in μs) verwendet werden. Die Verwendung eines zeitgesteuerten „Begin Motion at defined Time“-Kommandos bietet den Vorteil, dass sich die Übertragungszeit des Kommandos nicht auf den Startzeitpunkt der PTT auswirkt. Der Startzeitpunkt der PTT muss nur so gewählt werden, dass er zeitlich gesehen um einiges später eintritt, als man für die Übertragung des BT-Kommandos zu allen verteilten Systemen benötigt. In Abbildung 5.3 ist der zeitgesteuerte Start der PTT mit dem Simple-IQ-Kommando „Begin Motion at defined Time“ grafisch dargestellt.

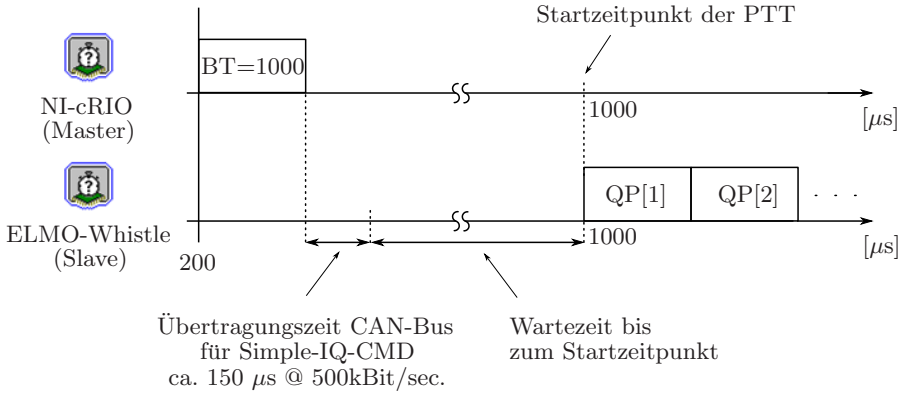


Abbildung 5.3: Start der PTT über das Kommando „Begin Motion at defined Time“ (BT)

5.2 Synchronisation zwischen dem Master-cRIO und den Slave-cRIOs (Multiroboter-Szenario)

5.2.1 Wozu ist die Synchronisation zwischen den cRIOs notwendig?

Der Startzeitpunkt für den Fahrtbeginn eines (Multi)Roboterfahrwerkes wird über das Bedienterminal am Host-PC gesteuert, bei dem das Kommando „Fahrt starten“ an das Master-Roboterfahrwerk bzw. den Master-cRIO gesendet wird. Der Master-cRIO hat die Aufgabe, dieses Kommando auch an die beteiligten Slave-cRIOs bzw. Slave-Roboterfahrwerke zu senden. Damit der Fahrtbeginn aller Räder synchron erfolgt, muss das Kommando „Fahrt starten“ auf allen Roboterfahrwerken (cRIOs) zeitgleich ausgeführt werden. Um das zu erreichen, empfiehlt sich wieder auf die Verwendung eines zeitgesteuerten Kommandos zurückzugreifen, dessen Prinzip im Kapitel 5.1 erläutert wurde. Damit ein zeitgesteuertes Kommando auf mehreren verteilten Systemen synchron ausgeführt wird, ist es Voraussetzung, dass alle Systemuhren der verteilten Systeme synchronisiert wurden.

5.2.2 Verbindungstopologie - Netzwerkkommunikation

In Abbildung 5.4 ist die Verbindungs-Topologie zwischen dem Host-PC, dem Master-cRIO (Master-Roboterfahrwerk) und den übrigen Slave-cRIOs (Slave-Roboterfahrwerke) dargestellt. Alle cRIOs sind über Wireless-LAN miteinander verbunden und kommunizieren über das TCP/IP-Protokoll. Der Host-PC ist ein Rechner mit Windows-Betriebssystem auf dem die Pfadplanung für das (Multi)Roboterfahrwerk, die Visualisierung der Kinematik-Daten sowie die Bedienung des Multiroboter-Programms erfolgt. Auf das

Thema Multiroboter-Programm wird in Kapitel 6.2 noch detaillierter eingegangen, es soll in diesem Abschnitt nur eine grobe Übersicht über die Kommunikations-Struktur der einzelnen Teilnehmer erzeugt werden. Der Master-cRIO (Master-Roboterfahrwerk) spielt in der Kommunikation (siehe Abbildung 5.4) eine Hauptrolle, da er einen zentraler Punkt in der Kommunikationsstruktur ein nimmt und er auch für die Verwaltung aller Kommunikationsverbindungen verantwortlich ist.

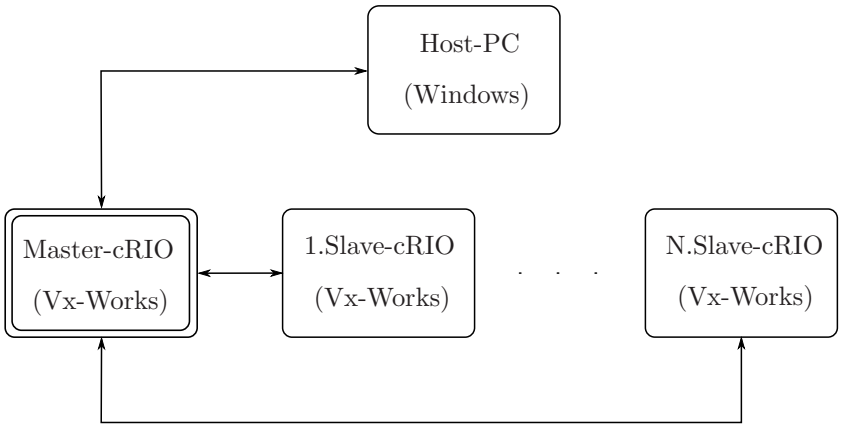


Abbildung 5.4: Netzwerkverbindungs-Topologie - Multiroboterszenario

Intuitiv würde es auf den ersten Blick logischer erscheinen, wenn der Host-PC diese Hauptrolle in der Kommunikation übernehmen würde bzw. der Host-PC im Zentrum der Kommunikationsstruktur stehen würde, da in diesem Falle die Daten der Slave-cRIOs nicht über den Master-cRIO geleitet werden müssten. Der Grund warum die Verbindungs-Topologie von 5.4 gewählt wurde, liegt in der Echtzeitfähigkeit der Systeme. Auf den NI-cRIOs läuft ein Vx-Works Echtzeit-Betriebssystem welches garantiert, dass das System eine Anfrage in einer vorhersehbaren bzw. berechenbaren Zeitdauer bearbeitet und anschließend eine Antwort liefert. Da der Host-PC auf einem Windows-Betriebssystem basiert, ist dieses System auch nicht echtzeitfähig. Das heißt, beim Host-PC kann nicht garantiert werden, dass er eine Anfrage in einem festgelegten Zeitintervall beantwortet, weil er zum Beispiel gerade mit einem anderen systemeigenen Prozess beschäftigt ist, den das Windows-Betriebssystem für wichtiger erachtet und andere Programme hinten anstehen müssen. Aufgrund der Echtzeitfähigkeit der beteiligten Systeme wurde die Netzwerkverbindungs-Topologie von Abbildung 5.4 gewählt. Somit ist gewährleistet, dass alle zeitkritischen Prozesse, wie die Kommunikation zwischen den cRIOs während der Fahrt des Multiroboterfahrwerkes, auf einem echtzeitfähigen System ausgeführt werden.

5.2.3 Übertragungs-Eigenschaften des WLAN-Netzwerkes

Prinzipiell könnte man den Synchronisationsprozess gleich gestalten wie beim CAN-Bus-Netzwerk. Der Master-cRIO sendet hier periodisch SYNC-Timestamp-Paare (siehe 5.1.1) an die Slave-Teilnehmer, was jedoch bei einem WLAN-Netzwerk nicht sehr sinnvoll wäre. Der Grund liegt darin, dass es sich beim CAN-Bus-Netzwerk aufgrund des verwendeten Bus-Zugriffsverfahren CSMA/CR (Carrier Sense Multiple Access/Collision Resolution) um ein hoch deterministisches Netzwerk handelt. Bei diesem Bus-Zugriffsverfahren wird durch eine Arbitrierungsphase [8] sichergestellt, dass jener Teilnehmer, der die prioritärste Nachricht senden möchte, auch das Bus-Zugriffsrecht bekommt. Je niederwärtiger der Identifier eines CAN-Telegramms ist, desto höher ist seine Sendepriorität. Für eine genaue Beschreibung des CAN-Bus-Systems, wird auf [8] verwiesen. Durch den niedrigen Identifier (0x80h) hat das SYNC-Kommando am CAN-Bus höchste Priorität. Dadurch ist gewährleistet, dass das vom Master gesendete SYNC-Kommando innerhalb von wenigen Microsekunden bei den anderen Slaves eintrifft. Durch die hohe Priorität spezieller Kommandos und die hohe Übertragungsgeschwindigkeit macht es Sinn, den Zeitstempel gesondert in einem zweiten Schritt zu übertragen. Mit dem SYNC-Timestamp-Verfahren können Teilnehmer bei einem CAN-Bus-Netzwerk bis auf wenige 10 μs genau synchronisiert werden.
Bei einem WLAN-Netzwerk wird als Übertragungsmedium die Atmosphäre verwendet. Damit sich die Funksignale der einzelnen Teilnehmer nicht gegenseitig stören, wurde auch hier ein spezielles Zugriffsverfahren CSMA/CA (Carrier Sense Multiple Access/Collision Avoidance) für die Teilnehmer entwickelt, welches verhindert, dass sich die Funksignale der einzelnen Teilnehmer bei der Datenübertragung stören. Bei dem Zugriffsverfahren CSMA/CA wird vereinfacht ausgedrückt durch „abhören“ des Übertragungsmediums sichergestellt, dass vor Sendebeginn eines Teilnehmers kein anderer das Übertragungsmedium belegt. Wenn doch, dann müssen alle anderen Teilnehmer die auch Daten übertragen möchten warten, bis das Übertragungsmedium wieder frei ist. Dabei wird die Wartezeiten der Teilnehmer zufällig über einen Algorithmus bestimmt. Nach Ablauf der Wartezeit startet jeder Teilnehmer erneut einen Sendeversuch, bei dem er das Übertragungsmedium wieder „abhört“ ob es belegt ist. Wenn es frei ist, dann bekommt der Teilnehmer das Zugriffsrecht und kann seine Daten übertragen. Aufgrund der zufällig bestimmten Wartezeiten und der Tatsache, dass bei dem Zugriffsverfahren CSMA/CA den Nachrichten auch keine Prioritäten zugeteilt werden können, sind Netzwerke die auf WLAN (IEEE 802.11) basieren auch nicht echtzeitfähig. Dadurch ist auch nicht vorhersehbar bzw. exakt

berechenbar, wann eine Datenübertragung über ein WLAN-Netzwerk abgeschlossen ist. Theoretisch betrachtet ist eine Synchronisation von mehreren Teilnehmern über ein nicht deterministisches Netzwerk nicht möglich, weil nicht vorhergesehen werden kann, wie lange für die Übertragung eines Master-Datenpaketes an die Slave-Teilnehmer benötigt wird. In der Praxis kann jedoch bei einem Ethernet- oder WLAN-Netzwerk folgende Annahme getroffen werden:

Bei der Verwendung eines Ethernet- oder WLAN-Netzwerkes sollte unter der Voraussetzung, dass die Netzwerkverbindung nur sehr schwach ausgelastet ist, eine Synchronisation der Netzwerk-Teilnehmer bis auf wenige ms möglich sein. Dabei kommt es durch die schwache Netzwerkauslastung nur zu wenigen oder keinen Kollisionen(Ethernet) oder Behinderungen(WLAN) bei der Übertragung der Datenpakete.

Die Voraussetzung für diese Annahme ist zum Beispiel erfüllt, wenn nur ein Teilnehmer das Übertragungsmedium für die Datenübertragung nutzt. In der Synchronisationsphase sendet nur der Master-Teilnehmer Datenpakete, wie zum Beispiel Zeitstempel, an die zu synchronisierenden Slave-Teilnehmer. In diesem Falle müsste eine Synchronisation der Teilnehmer über das WLAN-Netzwerk möglich sein. Ob sich diese Annahme mit der verwendeten Hard- und Software bewahrheitet, wird ein kleines Versuchsprogramm zeigen, bei dem die Übertragungs-Eigenschaften der WLAN-Netzwerkverbindung näher untersucht werden.

5.2.4 Ermittlung der Übertragungseigenschaften (WLAN-Netzwerkverbindung)

Für die Übertragung der Daten zwischen den cRIOs wird die „Simple Messaging Reference Library“ [1], kurz STM-Library von National Instruments verwendet. Diese Software-Bibliothek bietet dem Anwender die Möglichkeit, relativ einfach Daten über ein Ethernet- oder WLAN-Netzwerk, basierend auf dem TCP/IP-Protokoll, zu übertragen. Bei der STM-Library kann der Anwender auf vorgefertigte Software-Module zurückgreifen, die sich um die Verwaltung der Netzwerkverbindung und um das Senden und Empfangen von Daten kümmern. Der Anwender braucht nur mehr zu definieren, welchen Namen und Datentyp (Integer,Boolean,Cluster,...) die zu übertragenen Daten haben. Der Vorteil der STM-Library liegt darin, dass der Software-Entwickler über kein spezielles Fachwissen, betreffend Kommunikation mit TCP/IP-Protokollen, verfügen muss, um Daten

[1] http://zone.ni.com/devzone/cda/epd/p/id/2739

über ein Ethernet- oder WLAN-Netzwerk zu übertragen. Basierend auf der STM-Library wurde das LabVIEW-Projekt(Software-Projekt) „Check WLAN-Network“ erstellt, welches die Aufgabe besitzt, die kleinste Periodendauer $T_{d,min}$ zu ermitteln, bei der noch ein erfolgreicher Datenaustausch zwischen 2 Recheneinheiten (cRIOs) garaniert werden kann. In dieses LabVIEW-Projekt sind zwei cRIOs eingebunden. Am Master-cRIO läuft das „Daten-Sende-Programm“, das Daten über ein Unterprogramm namens „STM Write Msg.vi“ zyklisch an den Slave-cRIO sendet. Auf dem Slave-cRIO wird das „Daten-Empfangs-Programm“ ausgeführt, das zyklisch ein Unterprogramm names „STM Read Msg.vi“ ausführt, um die empfangenen Daten des Master-cRIOs einzulesen. Der Master-cRIO sendet dabei 10 mal den Wert einer Variablen an den Slave-cRIO, der bei jeder Iteration vom Master-cRIO inkrementiert wird. Es werden zwei Versuche durchgeführt, wobei die Periodendauer des Sende- und Empfangszyklus beim ersten Versuch 100 ms und beim zweiten Versuch 40 ms beträgt. Für die anschließende Auswertung der Programmlaufzeiten wird das Programm „Real-Time Exectution Trace Toolkit“ von National Instruments verwendet. Über dieses Programm kann nach Ausführung des zu untersuchenden Programms genau bestimmt werden, welche Programmteile zu welchem Zeitpunkt ausgeführt wurden und wie lange die Recheneiheit (cRIO) für die Ausführung benötigt hat.

5.2.4.1 1.Versuch: Ein Sende- und Einlesezyklus von 100 ms

Die Ergebnisse aus dem Real-Time Exectution Trace Toolkit sind in Abbildung 5.5 dargestellt. Die schwarzen strichlierten Balken entsprechen beim Master-cRIO einem Sendezyklus und beim Slave-cRIO einem Einlesezyklus. Solch ein strichlierter Balken gibt Auskunft, welche Programmteile bei der Ausführung eines Sende- oder Einlesezyklus beteiligt waren, und wie lange der cRIO zur Ausführung benötigt hat. Durch den kleinen Zoomfaktor stellen sich jetzt die sequenziell ausgeführten Programmteile eines Sende- oder Einlesezyklus als „Balken“ dar. Da diese Versuche zeigen sollen, ab welcher Zyklusdauer der Slave-cRIO nicht mehr in der Lage ist, die vom Master-cRIO gesendeten Daten einzulesen, spielen hier die Ausführzeiten der beteiligten Unterprogramme eines Sende- oder Einlesezyklus eine untergeordnete Rolle. Wie aus der Abbildung 5.5 entnommen werden kann, ist bei einem Sendezyklus von 100 ms der Slave-cRIO in der Lage, alle 100 ms die empfangenen Daten einzulesen.

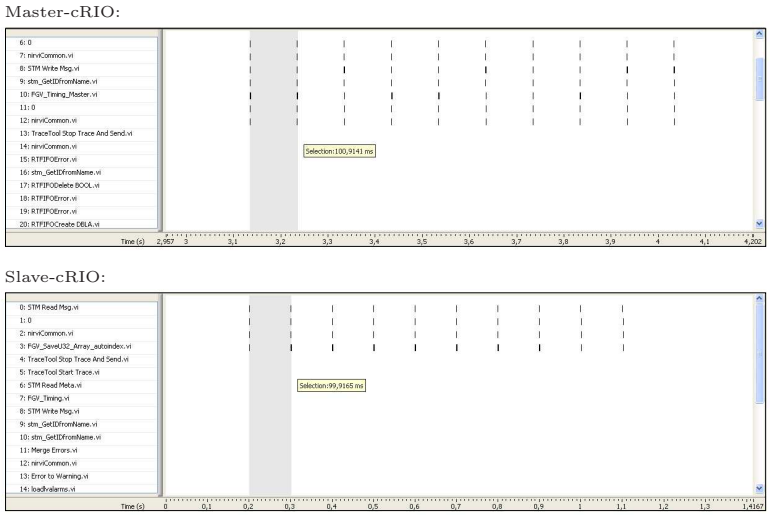


Abbildung 5.5: RT-Execution Trace Toolkit: Sende- und Empfangsperiode von 100ms

5.2.4.2 2.Versuch: Ein Sende- und Einlesezyklus von 40 ms

In Abbildung 5.6 sind die Ergebnisse aus dem Real-Time Exectution Trace Toolkit bei einer Zyklusdauer von 40 ms dargestellt. Es ist zu erkennen, dass der Master-cRIO alle 40 ms seine Daten an den Slave-cRIO sendet. Der Slave-cRIO ist jedoch nicht mehr in der Lage, alle 40 ms die Daten einzulesen. Das 3. und 7. Datenpaket wird mit einer Verzögerung von einer Zyklusdauer eingelesen.

Abbildung 5.6: RT-Execution Trace Toolkit: Sende- und Empfangsperiode von 40ms

In Abbildung 5.7 sind die CPU-Prozesse und Ausführzeitpunkte der Unterprogramme des

Master-cRIO-SendeProgramms, für den 2. und 3. Sendezyklus dargestellt. In dieser Detailansicht ist zu erkennen, dass beim 2. Sendezyklus, nachdem das Unterprogramm „STM Write Msg.vi“ ausgeführt wurde, im Anschluss von der CPU der „tNetTask“ ausgeführt wird. Der Slave-cRIO ist beim 2. Sendezyklus noch in der Lage, die gesendeten Daten einzulesen. Wir nun der 3. Sendezyklus betrachtet, so ist im Vergleich zum 2.Sendezyklus zu erkennen, dass hier nach dem das Unterprogramm „STM Write Msg.vi“ ausgeführt wurde, der „tNetTask”von der CPU nicht mehr ausgeführt wird. Der Slave-cRIO ist beim 3. und 7. Sendezyklus nicht mehr in der Lage, die Daten einzulesen. Erst mit der Verspätung von einer Zyklusdauer werden die Daten vom Slave-cRIO gelesen.

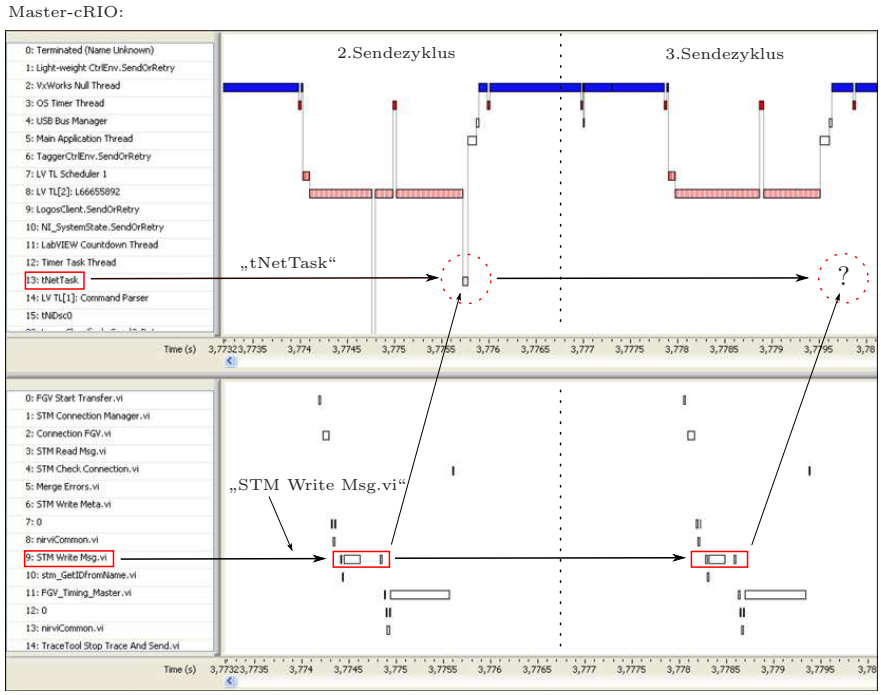


Abbildung 5.7: Datentransfer Master-cRIO (2. und 3. Datenpaket)

5.2.4.3 Fazit - Übertragungseigenschaften WLAN

Aus dem Versuch 2 (5.2.4.2) kann abgeleitet werden, dass für eine erfolgreiche Datenübertragung der „tNetTask“ eine Hauptrolle spielt. Wird nach der Ausführung des UnterProgramms „STM Write Msg.vi“ von der CPU der „tNetTask“ aufgerufen, dann werden die Daten vom cRIO per TCP/IP-Protokoll gesendet. Wird jedoch der „tNetTask“, nachdem die zu übertragenen Daten an das Unterprogramm „STM Write Msg.vi“ übergeben wurden, nicht ausgeführt, dann werden die Daten auch nicht an

den anderen cRIO übertragen. Leider konnte die Frage nicht geklärt werden, wovon es abhängt, ob der „tNetTask“ von der CPU ausgeführt wird oder nicht. Zusammenfassend kann jedoch gesagt werden, dass bei der Übertragung von Daten über ein Ethernet- oder WLAN-Netzwerk mittles der STM-Library, Sendezykluszeiten von 100 ms nicht unterschritten werden sollten, wenn die Daten ohne Verzögerung am Empfangs-cRIO eingelesen werden müssen. Damit gilt die getroffene Annahme von 5.2.3 für die hier verwendete Soft- und Hardware-Kombination (NI-cRIO mit SEA-WLAN-Modlus und STM-Library) nicht.

5.2.5 Synchronisations-Verfahren

In diesem Abschnitt sollen 3 Möglichkeiten vorgestellt werden, wie die Systemuhren der Slave-cRIOs an die des Master-cRIOs synchronisiert werden können. Dabei sind alle Teilnehmer (cRIOs) über WLAN miteinander verbunden. Es soll hier noch angemerkt werden, dass das Synchronisationsergebnis natürlich auch von der benötigten Netzwerk-Hardware wie Router und Switches abhängig ist. Im Anschluss soll dann überprüft werden, ob sich diese 3 vorgestellten Synchronisations-Verfahren mit der gegebenen Hardware 2 umsetzen lassen.

5.2.5.1 Software-Synchronisation mittels Master-Timestamp

Bei diesem Synchronisations-Verfahren sendet der Master-cRIO während der Synchronisationsphase zyklisch den Zeitstempel seiner Systemuhr an die zu synchronisierenden Slave-cRIOs. Die Slave-cRIOs sind aufgrund des empfangenen Master-cRIO-Zeitstempels in der Lage, ihre Systemuhren an die des Masters zu synchronisieren. Das Synchronisationsergebnis hängt bei diesem Verfahren von drei Zeitfaktoren ab. Der erste Faktor ist die Sendedauer, jene Zeitdauer die ab der Übergabe des Master-Zeitstempels an die Übertragungs-Software bis zum Beginn der physikalischen Datenübertragung vergeht. Der zweite Faktor ist die Übertragungsdauer, jene Zeitdauer die vergeht, bis die physikalische Datenübertragung des Zeitstempels über das Netzwerk zu den Slaves abgeschlossen ist. Der dritte Faktor ist die Einlesedauer, jene Zeitdauer, die das Daten-Einleseprogramm des Slaves benötigt, bis es den Master-Zeitstempel eingelesen hat. In Abbildung 5.8 sind diese Zusammenhänge nochmals grafisch dargestellt.
In dem Beispiel von Abbildung 5.8 hat die Systemuhrzeit des Salves einen Offset von 100 ms im Vergleich zur Master-Systemzeit. Mit Hilfe des gesendeten Master-Zeitstempels können die Slaves die Differenz ihrer Systemuhren bezogen auf die Master-Systemuhr er-

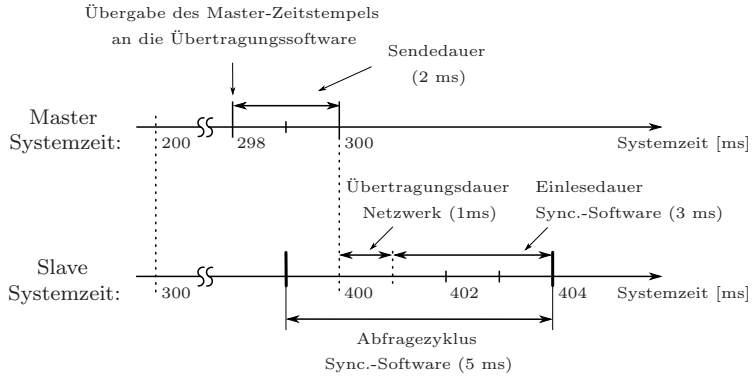


Abbildung 5.8: Synchronisationsbeispiel - Master Timestamp

mitteln. Die Daten des Master-Zeitstempels werden in diesem Beispiel zum Zeitpunkt 298 ms (absolute Master-Systemzeit) dem Master-Datenübertragungsprogramm übergeben. Die Daten werden dann zum Zeitpunkt 300 ms über das WLAN-Netzwerk gesendet. Die Übertragungsdauer wird mit 1 ms angenommen. Damit der Slave auf die Daten des gesendeten Zeitstempels in seinem Synchronisations-Programm zugreifen kann, muss er das Daten-Einleseprogramm zyklisch abfragen, ob neue Daten eingetroffen sind. Je schneller der Abfragezyklus ist, desto geringer ist die Einlesedauer. Die Einlesedauer wird für dieses Beispiel mit 3 ms festgelegt. Um den Offset der Slave-Systemuhr bezogen auf die Master-Systemuhr zu bestimmen, zieht der Slave den empfangenen Master-Zeitstempel von seiner aktuellen Systemzeit ab. Dadurch kann er die ungefähre Differenz zwischen seiner Systemuhr und der des Masters berechnen. In dem Beispiel von Abbildung 5.8 ergibt sich somit für den Slave eine Differenz von 106 ms (404ms-298ms = 106ms) bezogen auf die Mastersystemuhr. Durch die Sendedauer, die Übertragungszeit und die Einlesedauer ergibt sich ein Synchronisationsfehler der nicht korrigiert werden kann. In diesem Beispiel von 6 ms. Um den Synchronisationsfehler möglichst klein zu halten, sollte der Abfragezyklus der Daten-Empfangssoftware auf dem Slave-cRIO während der Synchronisationsphase so klein wie nur möglich sein. Durch diese Maßnahme verringert sich auch die Einlesedauer des Daten-EmpfangsProgramms. Die Übertragungszeit des Master-Zeitstempels über das Netzwerk kann positiv beeinflusst werden, wenn während der Synchronisations-Phase keine anderen Netzwerkteilnehmer Daten über das Netzwerk senden, weil dadurch die Übertragung des Master-Zeitstempels gestört wird und sich somit die Übertragungszeit des Master-Zeitstempels verlängern würde. Mit der Synchronisationsmethode „Softwaresynchronisation mittel Master-Timestamp“ sollte über das WLAN-Netzwerk eine Synchronisation der Systemuhren bis auf ca. 10 ms möglich sein.

5.2.5.2 Software Synchronisation Beta Program

Das „Software Synchronisation Beta Program“ ist eine Beta-Software [2] die „National Instruments“ über seine Homepage [3] (NI Developer Zone) zur Verfügung stellt. Mit dem Attribut „Beta“ soll verdeutlicht werden, dass sich diese Software noch in der Entwicklungsphase befindet und daher die Ausführsicherheit und Fehlerfreiheit dieses Programms noch nicht garantiert wird. Das „Software Synchronisation Beta Program“ verwendet den IEEE 1588 V2 Standard für die Synchronisation mehrerer LabVIEW Real-Time-Targets (auch cRIOs) über ein Ethernet- oder WLAN-Netzwerk. Der IEEE 1588 Standard, der auch als „Precision Clock Synchronisation Protocol for Networked Measurement and Control Systems“ oder abgekürzt PTP bezeichnet wird, definiert ein Verfahren, bei dem eine größere Anzahl von räumlich verteilten Echtzeituhren synchronisiert werden können, die über ein paketfähiges“ Netzwerk (Ethernet oder auch WLAN) miteinander verbunden sind. Dazu wird auf allen zu synchronisierenden cRIOs ein Synchronisations-Software-Paket installiert, das die Synchronisation der cRIO-Systemuhren nach dem IEEE 1588 Standard durchführt. Dadurch hat der Programmierer die Möglichkeit, sich bei der Verwendung von zeitgesteuerten Funktionen (Timed Loops,...) entweder auf eine absolute synchronisierte Systemzeit oder auf eine lokale Systemzeit des cRIO-Systems zu beziehen. Laut Programmbeschreibung soll es mit dieser Software möglich sein, die Systemuhren der NI-cRIOs bis auf wenige Millisekunden genau zu synchronisieren. Auf eine genauere Beschreibung des IEEE 1588 Standards wird in dieser Arbeit verzichtet, da dies für die weiteren Betrachtungen nicht notwendig ist.

5.2.5.3 Synchroner Boot-Up der cRIO-Systeme mittels gleicher Versorgungsspannung

Eine unkonventionelle Methode die Systemuhren aller cRIOs gleichzeitig zu starten, geht über deren Versorgungsspannung. Voraussetzung für diese Methode ist jedoch, dass es sich bei den verwendeten Recheneinheiten (cRIOs) um die selbe Hardware handelt. Ist diese Forderung erfüllt, dann müssen alle cRIOs an die selbe Versorgungsspannung angeschlossen werden. Damit ist gewährleistet, dass alle Recheneinheiten zum selben Zeitpunkt eingeschaltet werden. Durch diesen synchrone Boot-Up aller cRIO-Systeme kann angenommen werden, dass die Hardware-Systemuhren nahezu zeitgleich starten. Somit kann bei dieser Methode auf eine aktive Synchronisation der cRIO-Systemuhren über ein

[2] http://zone.ni.com/devzone/cda/tut/p/id/8278
[3] http://www.ni.com/devzone

Netzwerk verzichtet werden. Durch den synchronen Boot-Up der Systeme laufen deren Systemuhren schon nahezu synchron. Es soll hier vermerkt werden, dass es sich bei dieser „Synchronisations-Methode“ eher um eine Notlösung handelt, als um eine saubere Synchronisations-Methode.

5.3 Fazit - Synchronisationsmöglichkeiten

Damit mit der Methode „Software Synchronisation mittels Master-Timestamp“ eine Synchronisation der cRIO-Systemuhren im Bereich kleiner 10 ms gelingt, sollten die Slave-cRIOs das Daten-Einleseprogramm mir einer Periodendauer von ca. 5 ms abfragen (pollen). Diese Forderung konnte leider mit der STM-Library und der verwendeten Hardware nicht erreicht werden. Wie Untersuchungen aus Punkt 5.2.4 gezeigt haben, treten bei Sende- und Empfangszykluszeiten von kleiner als 50 ms schon Empfangsproblem auf. Wie erwähnt wurde, verwendet die STM-Library für die Kommunikation das verbindungsorientierte TCP/IP-Protokoll. Verbindungsorientiert heißt, das der Sender seine Daten erst an den Empfänger sendet, wenn dieser dazu bereit ist. Hierfür sendet der Empfänger eine Bereitschafts-Bestätigung an den Sender, der erst nach Empfang dieser mit der Datenübertragung beginnt. Diese Prozedur wird auch „Handshake“ genannt und garantiert eine sichere Datenübertragung, was jedoch zu Lasten der Daten-Übertragungsgeschwindigkeit geht. Eine schnellere Alternative zum TCP/IP-Protokoll wäre das UDP-Protokoll. Dieses Protokoll arbeitet nicht verbindungsorientiert, wodurch eine sichere Datenübertragung nicht gewährleistet werden kann. Durch den Verzicht an Übertragungssicherheit gewinnt man jedoch an Übertragungsgeschwindigkeit, was speziell in der Synchronisationsphase wichtig wäre. Mit der getroffenen Annahme aus Punkt 5.2.3 sollte die fehlende Übertragungssicherheit des UDP-Protokolles jedoch keine größeren Probleme bereiten, da in der Synchronisationsphase nur der Master-cRIO das Netzwerk belegt. Aus Zeitgründen konnte diese alternative Methode leider nicht mehr ausprobiert werden.

Die Synchronisation der cRIO-Systemuhren mittels der Software „Software Synchronisation Beta Program“ basierend auf den IEEE 1588 Synchronisationsverfahren konnte nach Anleitung installiert werden, jedoch konnte keine Synchronisation der cRIO-Systemuhren festgestellt werden. Die genaue Ursache für den nicht erfolgten Synchronisations-Vorgang konnte bis zum Abschluss dieser Arbeit nicht ermittelt werden. Da es sich um eine Beta-Software-Version handelt und eine einwandfreie Funktion

nicht garantiert wird, wurde auf eine zeitaufwändige Untersuchung des Problems verzichtet, da die Erfolgsaussichten eher als gering eingestuft wurden. Es wird für sinnvoller erachtet, in Zukunft ein Softwareversion des „Software Synchronisation Beta Programs“ zu verwenden, bei der es sich um keine Beta-Version mehr handelt. Dadurch sollte eine einwandfreie Funktion auch sichergestellt sein und eine Synchronisation der cRIO-Systemuhren möglich sein.

Da die ersten beiden Synchronisationsmethoden (Punkt 5.2.5.1 und Punkt 5.2.5.2) nicht von Erfolg gekrönt waren, musste leider auf die unkonventionelle Methode der Synchronisation über den zeitgleichen Boot-Up der cRIO-Systeme mittels gleicher Versorgungsspannung (Punkt 5.2.5.3) zurückgegriffen werden. Es wird jedoch empfohlen, diese Methode nur solange zu verwenden, bis eine der ersten beiden Synchronisations-Methoden erfolgreich umgesetzt werden kann, da in diesem Fall die Roboterfahrwerke immer mit einem Kabel verbunden werden müssen.

Kapitel 6

Software

6.1 Programmiersprache LabVIEW

Das Multiroboterprogramm „MPRD V3.4“ (Abschnitt 6.2) wurde mittels der Programmiersprache „LabVIEW“ von National Instruments[1] programmiert. LabVIEW ist eine grafische und datenflussorientierte Programmiersprache. Dabei kann der Programmierer unterschiedliche Funktionsblöcke (Summierer, Addierer, ...) auswählen und diese geeignet miteinander Verbinden, bis die gewünschte logische Funktion des LabVIEW-Programms gegeben ist. Programme werden in LabVIEW als VIs bezeichnet und Unterprogramme infolgedessen als Sub-VIs. Ein LabVIEW-VI besteht aus zwei Programmteilen, einem so genannten Front Panel und einem Blockdiagramm. Das Front Panel des LabVIEW-VIs dient dazu, die grafische Oberfläche des Programms festzulegen. Dabei können Anzeigeelemente (Graphen, Displays,...) und Bedienelemente (Schalter, Drehknöpfe,...) am Front Panel platziert werden. Im Blockdiagramm wird die logische Funktion des LabVIEW-VIs, durch die Verschaltung von unterschiedlichen Funktionsblöcken festgelegt. Jedes LabVIEW-VI kann wiederum in einem andern LabVIEW-VI aufgerufen werden. Dabei wird das aufgerufene LabVIEW-VI mit seinem definierten „Icon“ dargestellt, das über entsprechende Datenein- und Datenausgänge an den Seiten des Icons verfügt (siehe Abbildung 6.1). Über diese Schnittstellen erfolgt der Datenaustausch mit anderen LabVIEW-VIs. Zur Verdeutlichung dieses Zusammenhanges ist in Abbildung 6.2 das LabVIEW-Beispielprogramm „Main.vi“ mit dem integrierten Unterprogramm „RootMeanSquare.vi“, mit Front Panel und Blockdiagramm, dargestellt. Das Programm „Main.vi“ hat die Aufgabe, den quadratischen Mittelwert der Einstellknöpfe „Value 1“ und „Value 2“ zu berechnen. Dabei ist

[1] www.ni.com

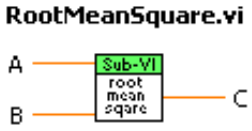


Abbildung 6.1: LabVIEW - Icon des Programms „RootMeanSquare.vi“

die Berechnungsvorschrift für den quadratischen Mittelwertes im Unterprogramm (Sub-VI) „RootMeanSquare.vi“ festgelegt. Das Programm „Main.vi“ übergibt dabei die beiden numerischen Werte der Einstellknöpfe „Value 1“ und „Value 2“ an das Unterprogramm „RootMeanSquare.vi“ und bekommt nach der Ausführung dieses Sub-VIs, das Ergebnis über die Ausgänge des Icons übermittelt. Der berechnete quadratische Mittelwert wird am Frontpanel des „Main.vi“, am Zeigermessgerät und in einer numerischen Anzeige, grafisch dargestellt.

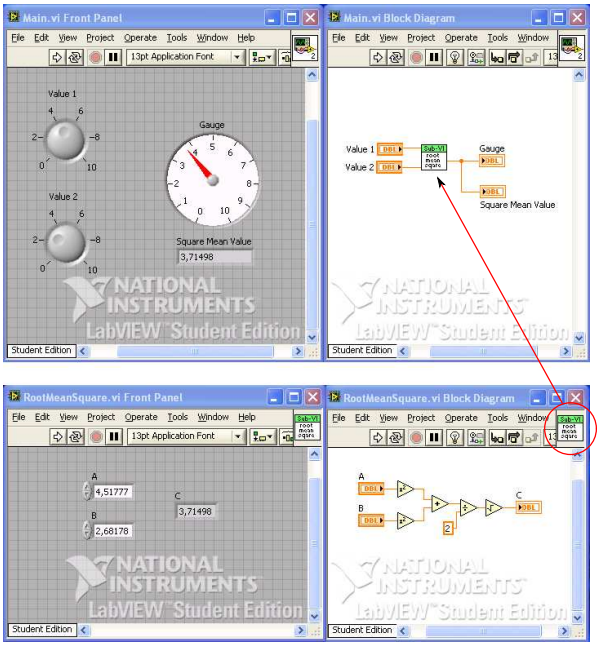


Abbildung 6.2: LabVIEW - Beispielprogramm „Main.vi“

6.2 Multiroboter-Programm

Das Multiroboter-Programm „MPRD V3.4“ besteht aus den LabVIEW-VIs „Host-Main.vi“, „Master-MainDriveComb.vi“ und „Slave-MainDriveComb.vi“. Diese drei Programme werden auf verschieden Systemen ausgeführt und erfüllen unterschiedliche

Funktionen im Multiroboter-Programm. Das Programm „Host-Main.vi“ wird auf dem Hauptcomputer (Host-PC) ausgeführt und dient als Bedien- und Anzeigeterminal für das Multiroboter-Fahrwerk. Das heißt, vom Front Panel des „Host-Main.vi“ aus können vom Benutzer Kommandos wie „cRIOs Initialisieren“ oder „Fahrt starten“ an das Multiroboter-Fahrwerk gesendet werden. Über das „Host-Main.vi“ wird auch der abzufahrenden Sollpfad für das Multiroboter-Fahrwerk festgelegt. Neben der Pfadplanung dient das „Host-Main.vi“ auch zur Visualisierung von Antriebs- und Positionsdaten, die während der Fahrt des Multiroboter-Fahrwerkes aufgenommen wurden. Das Programm „Master-MainDriveComb.vi“ wird, wie schon aus dem Namen ableitbar ist, auf dem cRIO des Masterroboterfahrwerkes ausgeführt. Es ist unter anderem für die Netzwerk-Verbindungsverwaltung, für die Weiterleitung der empfangenen Host-Kommandos an die Slaveroboterfahrwerke, für die Berechnung des Multiroboter-Bewegungsraumes $\mathbf{B}_m$ und für die Überprüfung der Ausführbarkeit der Multiroboter-Fahrbefehle $\dot{\boldsymbol{\xi}}_m$ zuständig. Sollte ein Fahrbefehl $\dot{\boldsymbol{\xi}}_m$ vom Multiroboter-Fahrwerk nicht ausgeführt werden können, dann wird eine mögliche Rekonfiguration (siehe Punkt 4.1.10) des Fahrbefehles $\dot{\boldsymbol{\xi}}_m$ durchgeführt. Das Programm „Slave-MainDriveComb.vi“ wird auf allen Slaveroboterfahrwerken des Multiroboter-Fahrwerkes ausgeführt. Dieses Programm wurde in der Multiroboter-Programmversion „MPRD V3.4“ so aufgebaut, dass es zwischen zwei Bewegungszenarien des Slaveroboterfahrwerkes unterscheiden kann. Im ersten Fall ist das Slaveroboterfahrwerk mir dem Masterroboter-Fahrwerk mechanisch verbunden oder bewegt sich synchron zum Masterroboter-Fahrwerk. Im zweiten Fall bewegt sich das Slaveroboterfahrwerk unabhängig vom Masterroboter-Fahrwerk. In diesem Fall muss für das Slaveroboterfahrwerk auch ein eigener Sollpfad vorgegeben werden. Die Berechnungsunterschiede in den beiden Bewegungsszenarien werden im Punkt 6.2.4 noch genauer ausgeführt.

6.2.1 Netzwerkkommunikation mittels STM-Library

Für die Datenübertragung zwischen Host-PC, Master-cRIO und den Slave-cRIOs wird die Software-Bibliothek „STM-Library“ von National Instruments verwendet. Diese Software-Bibliothek ist standardmäßig nicht in der LabVIEW-Programmiersoftware integriert, sondern muss nachträglich installiert werden. Die STM-Library kann über die Homepage[2] von National Instruments kostenlos bezogen werden. Diese Software-Bibliothek bietet dem Programmierer die Möglichkeit, Daten relativ einfach über ein Ethernet- oder

[2] http://zone.ni.com/devzone/cda/epd/p/id/2739

WLAN-Netzwerk, basierend auf dem TCP/IP-Protokoll, zu übertragen. Bei der STM-Library kann der Programmierer auf vorgefertigte Software-Module zurückgreifen, die sich um die Verwaltung der Netzwerkverbindung und um das Senden und Empfangen von Daten kümmern. Der Programmierer muss nur mehr definieren, welchen Namen (Sendevariable oder Meta Element) und Datentyp (Integer,Boolean,Cluster,...) die zu übertragenen Daten haben. Ein Nachteil dieser STM-Library ist, dass der direkte Datenaustausch zwischen zwei Slave-Netzwerkteilnehmern, zum Beispiel zwischen Host-PC und einem Slaveroboterfahrwerk oder umgekehrt, nicht möglich ist. Aufgrund dieser Einschränkung müssen die Daten zwischen zwei Slave-Netzwerkteilnehmern immer über den Master-Netzwerkteilnehmer (Master-cRIO) geleitet werden. Im Folgenden werden die wichtigsten Programme (VIs) der STM-Library kurz erklärt.

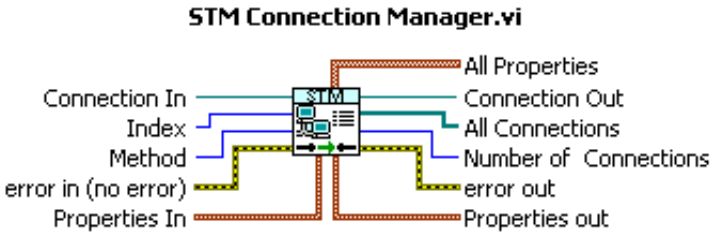


Abbildung 6.3: Icon „STM Connection Manager.vi“

STM Connection Manager.vi:
Dieses VI ist eines der wichtigsten Programme der STM-Library. Im „STM Connection Manager.vi“ werden die aufgebauten Netzwerkverbindungen verwaltet. Jede Netzwerkverbindung besitzt eine Verbindungs-ID (Connection-ID), die für jeden Netzwerkteilnehmer im Programm „STM Connection Manager.vi“ gespeichert wird. In der Tabelle 6.1 sind die Programmfunktionen aufgelistet, die das VI „STM Connection Manager“ anbietet. Diese werden über den Icon-Eingang „Method“ ausgewählt.

Method	**Funktion**
Initialize	Löscht alle internen Speicher und Variablen.
Set Properties	Verbindungsspezifische Daten können einer Netzwerkverb. zugewiesen werden.
Get Connections	Auslesen aller Netzwerkverbindungs-IDs.
Close Connection	Beenden einer Netzwerkverbindung.
Close Connections	Beenden aller Netzwerkverbindungen.

Tabelle 6.1: Funktionen des „STM Connection Manager.vi“

Abbildung 6.4: Icon „STM Check Connection.vi“

STM Check Connection.vi:
Über dieses VI wird überprüft, ob die ausgewählte Netzwerkverbindung noch besteht. Wenn nicht, dann wird die Verbindung im „STM Connection Manager“ geschlossen. Damit wird verhindert, das Netzwerkverbindungen, die einseitig getrennt wurden und daher nicht im „STM Connection Manager.vi“ geschlossen wurden, weiterhin vom Master-Netzwerkteilnehmer auf Dateneingang abgefragt werden.

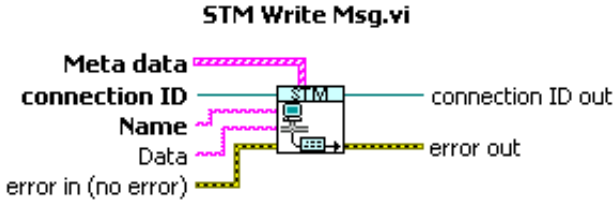


Abbildung 6.5: Icon „STM Write Msg.vi“

STM Write Msg.vi:
Über dieses Programm werden Daten entweder vom Master-Netzwerkteilnehmer an einen Slave-Netzwerkteilnehmer oder von einem Slave-Netzwerkteilnehmer an den Master-Netzwerkteilnehmer gesendet. Dem „STM Write Msg.vi“ wird der Name (Meta Element) der zu übertragenen Daten übergeben, wobei dieser schon im Vorfeld in den „Meta Daten“ (Sendevariablen) des Master-Netzwerkteilnehmers definiert werden muss. Über den „Data“-Eingang des VIs können die zu übertragenen Daten an den gewünschten Empfänger, mit zugehöriger „Connection ID“, gesendet werden. Die „Connection ID“ kann dabei vom Master-Netzwerkteilnehmer über das VI „STM Connection Manager“ (Method: Get Connection) bezogen werden. Die Slave-Netzwerkteilnehmer erhalten die Verbindungs-IDs vom Master-Netzwerkteilnehmer im Zuge des Verbindungsaufbaues. Es soll hier nochmals angemerkt werden, dass die Slave-Netzwerkteilnehmer nur Daten an den Master-Netzwerkteilnehmer senden können.

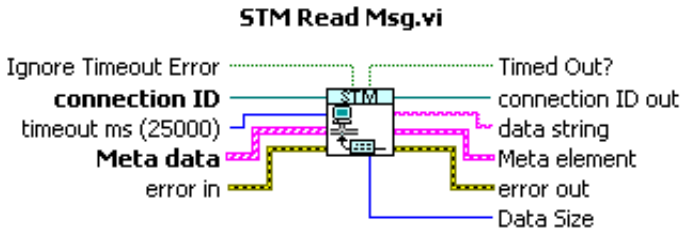


Abbildung 6.6: Icon „STM Read Msg.vi“

STM Read Msg.vi:
Mit Hilfe dieses Programms kann ein Netzwerkteilnehmer abfragen, ob Daten an ihn gesendet wurden. Wenn das der Fall ist, dann liefert der Ausgang „Timed Out?“ des „STM Read Msg.vi“ den bollschen Wert „False“. In weiterer Folge kann der Empfänger den Datentyp der empfangenen Daten über das zugeordnete „Meta element“ (Sendevariable) bestimmen und den empfangenen Daten-String richtig decodieren.

6.2.1.1 Chronologie des Netzwerkverbindungsaufbaus

Aufgrund der Programmstruktur der STM-Library muss als erstes der Master-Netzwerkteilnehmer gestartet werden. Im Multiroboter-Programm „MPRD V3.4“ ist das der Master-cRIO mit dem dazugehörigen Programm „Master-MainDriveComb.vi“. Nachdem dieses Programm gestartet wurde, können die Programme der Slave-Netzwerkteilnehmer (Host-PC und Slaveroboterfahrwerke) gestartet werden. Es ist jedoch zu beachten, dass aufgrund von programmtechnischen Gründen zuerst das Programm „Host-Main.vi“ und dann die Programme „Slave-MainDriveComb.vi“ zu starten sind. Wenn das Programm „Master-MainDriveComb.vi“ am Master-cRIO gestartet wird, werden als erstes die „Meta Daten“ (Sendevariablen) , die im Programm statisch gespeichert sind, geladen. In Abbildung 6.7 sind jene Programme in einem Datenflussdiagramm dargestellt, die im Programm „Master-MainDriveComb.vi“ beim Netzwerkverbindungsaufbau aufgerufen werden.
Nachdem die Meta Daten geladen wurden, wird das Programm „STM Connection Manager.vi“ initialisiert. Dabei werden die Inhalte alle internen Speicher und Schieberegister des Programms gelöscht. Im nächsten Schritt wird das Programm „TCP New Connection Monitor.vi“ ausgeführt. Dieses Programm hat die Aufgabe, Verbindungsanfragen von Slave-Netzwerkteilnehmern zu bearbeiten. Dabei hört das Programm „TCP Wait on Listner.vi“ einen im vorhinein definierten „Remote Port“ ab. Wird nun das Hauptprogramm eines Slave-Netzwerkteilnehmers gestartet, dann wird zuerst das

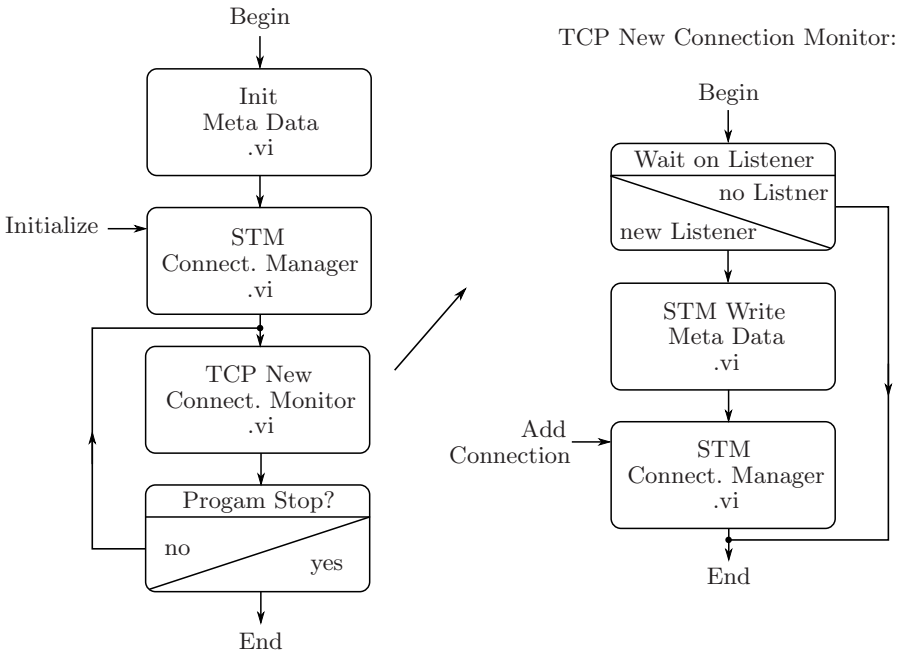


Abbildung 6.7: Datenflussdiagramm - Verbindungsaufbau „Master-Netzwerkteilnehmer“

Programm „TCP open Connection.vi“ ausgeführt. In Abbildung 6.8 sind die Programme grafisch dargestellt, die am Verbindungsaufbau eines Slave-Netzwerkteilnehmers beteiligt sind. Dem Programm „TCP open Connection.vi“ wird die IP-Adresse des Master-Netzwerkteilnehmers (Master-cRIO) und der „Remote Port“, an dem der Master-Netzwerkteilnehmer das Netzwerk auf neue Verbindungsanfragen abhört, übergeben. Diese Daten können auf der Recheneinheit eines Slave-Netzwerkteilnehmers gespeichert werden. Das Programm „TCP open Connection.vi“ erzeugt bei Aufruf einen „TCP Listner“ der vom Master-Programm „TCP Wait on Listner.vi“ registriert wird. In weiterer Folge bekommt der Slave-Netzwerkteilnehmer die Netzwerkverbindungs-ID (Connection-ID) für diese Master-Slave bzw. Slave-Master Netzwerkverbindung mitgeteilt. Nachdem der Slave-Netzwerkteilnehmer die Netzwerkverbindungs-ID erhalten hat, wartet er auf den Empfang der „Meta Daten“ (Sendevariablen), die er vom Master-Netzwerkteilnehmer im Zuge des Netzwerkverbindungsaufbaues übermittelt bekommt. Mit der Netzwerkverbindungs-ID (Connection-ID) und den „Meta-Daten“ (Sendevariablen) ist der Slave-Netzwerkteilnehmer nun in der Lage, Daten an den Master-Teilnehmer (Master-cRIO) zu senden. Am Master-Teilnehmer werden die Netzwerkverbindungs-IDs (Connection-IDs) für die Master-Slave Netzwerkverbindungen im Programm „STM Connection Manager.vi“ (Method:Add Connection) gespeichert.

Mit der beschriebenen Vorgangsweise, werden die Netzwerkverbindungen zu allen Slave-Netzwerkteilnehmern (Host-PC, Slaveroboterfahrwerke) aufgebaut. Verwaltet werden diese Netzwerkverbindungen vom Master-cRIO der auch die Funktion des Master-Netzwerkteilnehmers übernimmt.

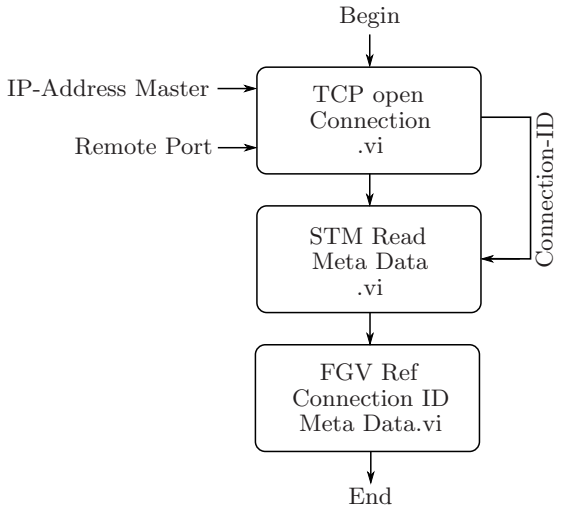


Abbildung 6.8: Datenflussdiagramm - Verbindungsaufbau „Slave-Netzwerkteilnehmer“

6.2.2 Programm „Host-Main.vi“

Abbildung 6.9: Icon des Host-Main.vi

Das LabVIEW-Programm „Host-Main.vi“ wird auf dem Host-PC ausgeführt und dient als Bedien- und Anzeigeterminal für das Multiroboter-Programm. Wenn das Programm „Host-Main.vi“ gestartet wird, werden nach dem Netzwerkverbindungsaufbau zum Master-cRIO, der im Punkt 6.2.1.1 beschrieben wurde, als erstes alle Programmvariablen gesetzt, die standardmäßig abgespeichert wurden (Default Values). Unter anderem wird auch der gespeicherte Standard-Sollpfad für das Multiroboter-Fahrwerk geladen. Der Anwender hat jedoch später noch die Möglichkeit, diesen Standard-Sollpfad durch einen anderen zu ersetzen oder einen neuen Sollpfad für das Multiroboterfahrwerk festzulegen. Nachdem die standardmäßig eingestellten Programmeinstellungen geladen wurde, werden vom „Host-Main.vi“ die Namen, IP-Adressen und Wabentypen der beteiligten Roboterfahrwerke eingelesen und auf dem Front Panel des „Host-Main.vi“, im Fenster

„Robot-Info“ (Registerkarte „Roboterkonfiguration“), angezeigt. In Abbildung 6.10 ist die Registerkarte „Roboterkonfiguration“ grafisch dargestellt.

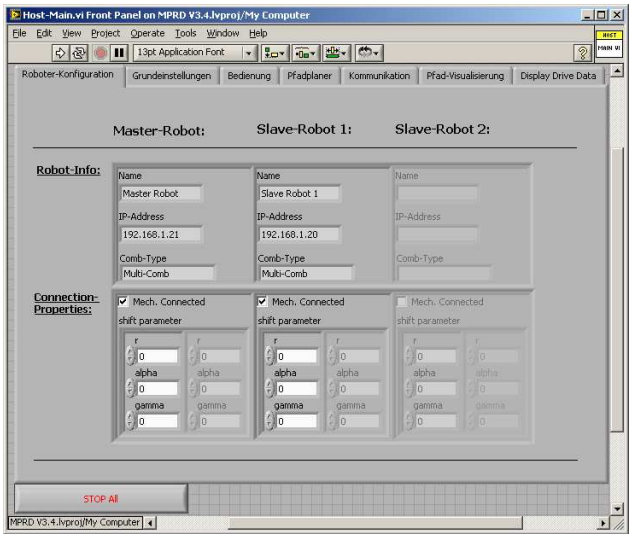


Abbildung 6.10: Registerkarte - Roboterkonfiguration

Unter dem Begriff „Waben-Typ“ eines Roboterfahrwerkes wird in der Multirobotersoftware „MPRD V3.4“ festgelegt, ob das Roboterfahrwerk aus einer (Single-Comb) oder aus mehreren Radwaben (Multi-Comb) aufgebaut ist. Diese Information wird bei der Berechnung der Roboterposition benötigt, die vom Masterroboterfahrwerk durchgeführt wird. Besteht das Masterroboterfahrwerk nur aus einer Radwabe (siehe Punkt 4.1.11), dann benötigt es noch zusätzlich die Antriebsdaten (Radlenkwinkel und Raddrehzahl) eines anderen verbundenen Slaveroboterfahrwerkrades, um die Position des Multiroboterfahrwerkes berechnen zu können. Im Fenster „Connection Properties“ (Registerkarte „Roboterkonfiguration“) wird vom Benutzer festgelegt, ob die Slaveroboterfahrwerke mit dem Masterroboterfahrwerk verbunden sind, was durch die Auswahl (Checkbox) „Mech. connected“ festgelegt wird. Diese Information dient dem Masterroboterfahrwerk dazu, ob es den Multiroboter-Fahrbefehl $\dot{\boldsymbol{\xi}}_m$ auch an die Slaveroboterfahrwerke senden muss oder nicht. Wenn zum Beispiel nur das Masterroboterfahrwerk den vorgegeben Sollpfad abfahren soll, dann braucht diese Checkbox nicht aktiviert zu werden. Im Multiroboterfall ist diese Checkbox jedoch bei jedem beteiligten Roboterfahrwerk zu aktivieren. Würde das im Multiroboterfall nicht gemacht werden, dann müsste ein eigener abzufahrender Sollpfad für das oder die Slaveroboterfahrwerke festgelegt werden. Diese Option ist jedoch in der aktuellen Multirobotersoftware „MPRD V3.4“ noch nicht vorgesehen. Zum anderen werden in den

„Connection Properties“ noch die Transformationsparameter (Shift-Parameter) für die lokalen Mittelpunkte der Roboterfahrwerke festgelegt. Diese werden benötigt, damit im Multiroboterfall die lokalen Robotermittelpunkte $\mathbf{p}_{0,i}$ der beteiligten Fahrwerke, in den festgelegten lokalen Multirobotermittelpunkt $\mathbf{p}_{0,m}$ verschoben werden können (siehe Punkt 4.1.8). Die Transformationsparameter (Shift-Parameter) müssen aus Sicht der lokalen Roboterkoordinatensysteme $\Sigma_{R,i}$ angegeben werden (siehe Abbildung).
Im nächsten Schritt müssen die Grundeinstellungen in der Registerkarte „Grundeinstellungen“ des „Host-Main.vi“ festgelegt werden. In Abbildung 6.11 ist diese Registerkarte grafisch dargestellt. Über die Auswahl „Gamepad-Steuerung“ oder „Pfad-Steuerung“ wird festgelegt, ob der Multiroboter-Fahrbefehl $\dot{\boldsymbol{\xi}}_m$ aus den Steuerbefehlen des Gamepads oder aus dem vorgegebenen Sollpfad für das Multiroboterfahrwerk abgeleitet wird. Neben der Steuerungsart wird auch noch die Berechnungszykluszeit T_d definiert. Über diese wird festgelegt, mit welcher Zykluszeit die Radlenkwinkel und Raddrehzahlwerte für die Roboterfahrwerke berechnet werden. Der Maximalwert der Zykluszeit von 100 ms wird durch die Motorregelgeräte (ELMO-Whistles) vorgegeben. Der Grund liegt in der beschränkten Anzahl (max. 256) von interpolierten Werten, die ein „ELMO-Whistles“ zwischen zwei Sollwertvorgaben einfügen kann. Wird die maximale Anzahl der Interpolationsstützstellen mit der Abtastperiode des ELMO-Whistle-Positionsregelkreises (400 μs) multipliziert, dann ergibt sich eine maximale Berechnungszykluszeit von 102,4 ms. Im Programm „Host-Main.vi“ wurde diese auf 100 ms abgerundet. Die untere Schranke der Berechnungszykluszeit T_d wird hauptsächlich durch die Dauer der Netzwerkkommunikation festgelegt. Aufgrund der angesprochenen Kommunikationsprobleme (siehe Punkt 5.2.4.3) wird jedoch bei der aktuellen Programmversion „MPRD V3.4“ empfohlen, dass die standardmäßig eingestellte Berechnungszykluszeit von 100 ms beibehalten werden sollte, da sonst der Datenaustausch zwischen den Roboterfahrwerken zusätzlich erschwert wird.
Neben der Berechnungszykluszeit T_d kann in der Registerkarte „Grundeinstellungen“ auch noch der Fahrmodus festgelegt werden, mit dem der Multiroboter-Sollpfad abgefahren werden soll. Besonders zu erwähnen sind hier der „Eyecatcher“ und der „Fixed Theta“-Fahrmodus. Beim „Eyecatcher“-Fahrmodus richtet sich das Muliroboterfahrwerk immer auf einen vorgegebenen Punkt in der Bewegungsebene aus, während es den definierten Sollpfad abfährt. Beim „Fixed Theta“-Fahrmodus behält das Roboterfahrwerk immer einen festgelegten Winkel Θ, zum raumfesten globalen Multiroboter-Koordinatensystem Σ_I ein. Eine ausführliche Beschreibung der Fahrmodi

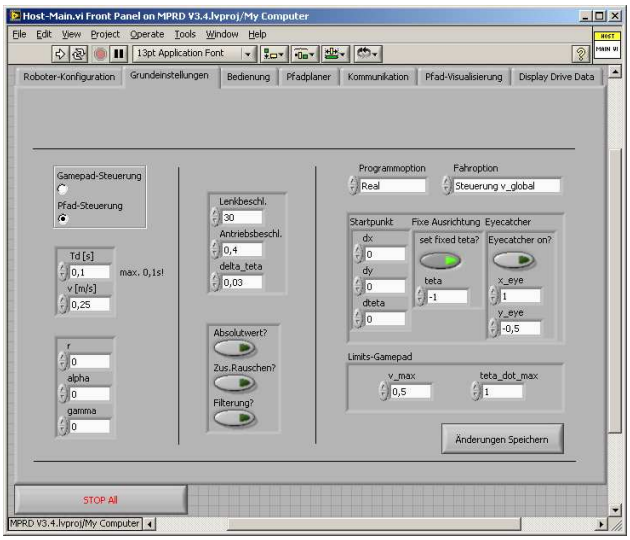


Abbildung 6.11: „Host-Main.vi“ - Registerkarte „Grundeinstellungen“

wird in [14] gegeben.

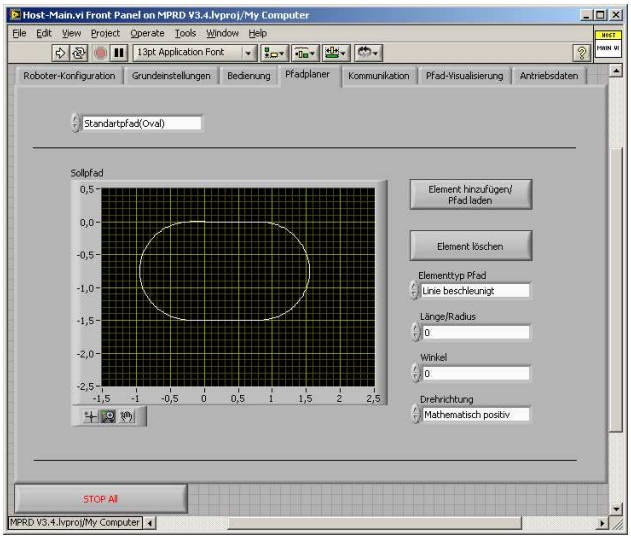


Abbildung 6.12: „Host-Main.vi“ - Registerkarte „Pfadplaner“

Über die Registerkarte „Pfadplaner“ des „Host-Main.vi“ kann der Benutzer den abzufahrenden Sollpfad für das Multiroboterfahrwerk festlegen. Dieser Sollpfad bezieht sich auf den Multirobotermittelpunkt $\mathbf{p}_{0,m}$ eines Multiroboterfahrwerkes.
Die Steuerung des Multiroboterfahrwerkes erfolgt über die Registerkarte „Bedienung“ (siehe Abbildung 6.13) des „Host-Main.vi“. Zu Beginn müssen die Recheneinheiten (cRIOs) der Roboterfahrwerke initialisiert werden. Das geschieht über die Schaltfläche „cRIOs

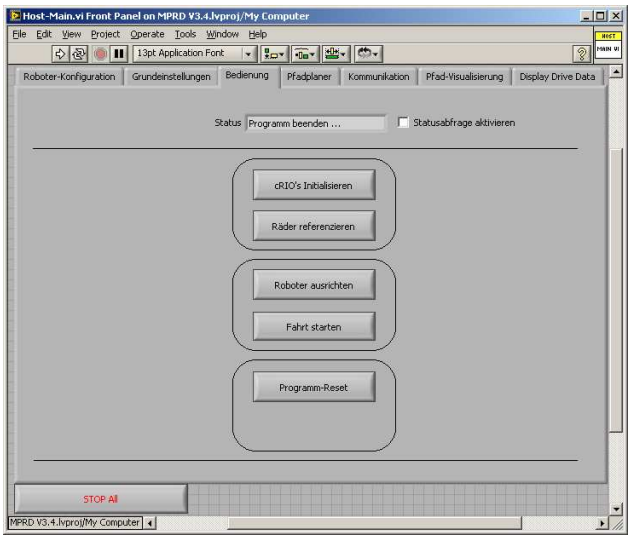


Abbildung 6.13: „Host-Main.vi“ - Registerkarte „Bedienterminal“

initialisieren“. Dabei werden unter anderem die Motorreglegeräte (ELMO-Whistle) der Radwaben parametriert, die Variablen und Datenspeicher der cRIOs initialisiert und im Multiroboterfall senden die Slaveroboterfahrwerke ihre Bewegungsräume $\mathbf{Z}_i$ und $\bar{\mathbf{S}}_i$ an das Masterroboterfahrwerk. Damit ist das Masterroboterfahrwerk bzw. das Programm „Master-MainDriveComb.vi“ später in der Lage, den Bewegungsraum $\mathbf{B}_m$ für das Multiroboterfahrwerk zu berechnen. Über die Schaltfläche „Räder referenzieren“ ermitteln die Radwaben aller Roboterfahrwerke die Lage ihrer Radlenkwinkel-Referenzposition, die durch eine Lichtschranke festgelegt ist. Im Anschluss werden die festgelegten Radlenkoffsetwinkel, die in der Modelldefinition der Roboterfahrwerke gespeichert sind, von den Radwaben-Lenkeinheiten angefahren. Nach der Referenzierung der Räder zeigen alle Radachsen eines Roboterfahrwerkes in Richtung Robotermittelpunkt $\mathbf{p}_{0,i}$ (siehe Punkt 4.1.9). Nach erfolgreicher Referenzierung der Radwaben-Lenkeinheit, kann das Multiroboter-Fahrwerk ausgerichtet werden. Das geschieht über die Schaltfläche „Roboter ausrichten“ und ist speziell bei den Fahrmodi „Eyecatcher“ und „Fixed Theta“ notwendig. In diesen Fällen muss das Multiroboter-Fahrwerk einen vorgegebenen Winkel Θ zum raumfesten globalen Roboterkoordinatensystem Σ_I einnehmen. Zusätzlich wird bei diesem Kommando vom Masterroboterfahrwerk der Multiroboter-Bewegungsraum $\mathbf{B}_m$ für das Multiroboter-Fahrwerk berechnet. Nachdem das Multiroboter-Fahrwerk ausgerichtet wurde, kann der Befehl für den Start der Pfadabfahrt erfolgen. Das geschieht über die Schaltfläche „Fahrt Starten“ der Registerkarte „Bedienung“. Nachdem das Kommando „Fahrt starten“ an alle Roboterfahrwerke gesendet wurde, fährt das

Multiroboterfahrwerk den vorgegebenen Sollpfad ab.
Wenn das Multiroboter-Fahrwerk den vorgegebenen Sollpfad erfolgreich abgefahren hat, werden im Anschluss die Positionsdaten des Multiroboter-Fahrwerkes, die während der Fahrt berechnet wurden, in der Registerkarte „Pfad-Visualisierung“ dargestellt. Diese globalen Positionsdaten werden mithilfe des Programms „Odometrie.vi“ aufgrund der Antriebsdaten (Radlenkwinkel und Raddrehzahl) der Roboterräder des Masterroboterfahrwerkes berechnet. Auf den Motorregelgeräten (ELMO-Whistle) der Radwaben, wird zwischen den einzelnen Sollwertvorgaben 3. Ordnung interpoliert. Dadurch ergibt sich zwischen der Vorgabe und dem Anfahren eines Sollwertes eine Verzögerung von 3 Berechnungsperioden T_d. Durch diese Verzögerung entsteht bei der Berechnung der globalen Roboterposition über das Programm „Odometrie.vi“, das aus dem Vorgängerprogramm [14] übernommen wurde, ein kleiner Fehler. Da die Position des Roboterfahrwerkes wiederum in die Berechnung des nächsten Radlenkwinkel- und Raddrehzahlsollwertes eingeht, ergibt sich während der Fahrt eine kleine Abweichung vom vorgegebenen Sollpfad.

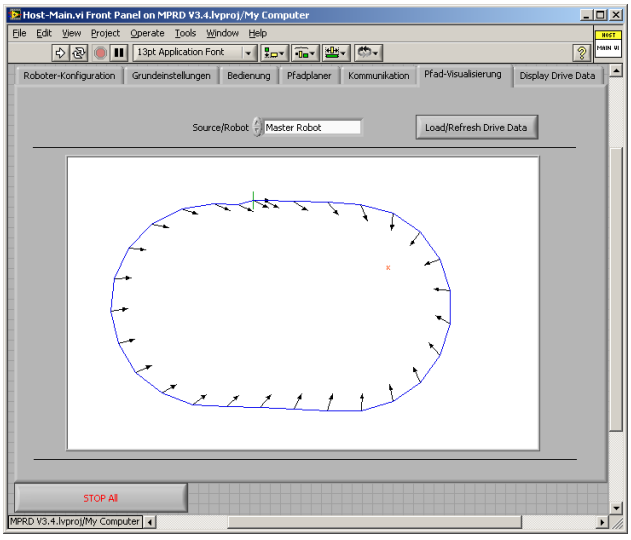


Abbildung 6.14: „Host-Main.vi“ - Registerkarte „Pfad Visualisierung“

Zusätzlich zur Pfadvisualisierung kann der Benutzer nach Beendigung der Multiroboterfahrt auch die Radlenkwinkel und Raddrehzahlwerte der einzelnen Multiroboterfahrwerksräder in Augenschein nehmen. Das geschieht über die Registerkarte „Antriebsdaten“ des „Host-Main.vi“. Über die Schaltfläche „Export Data to File“ können die Radlenkwinkel- und Raddrehzahldaten der Roboterfahrwerke in einer Text-Datei gespeichert werden.

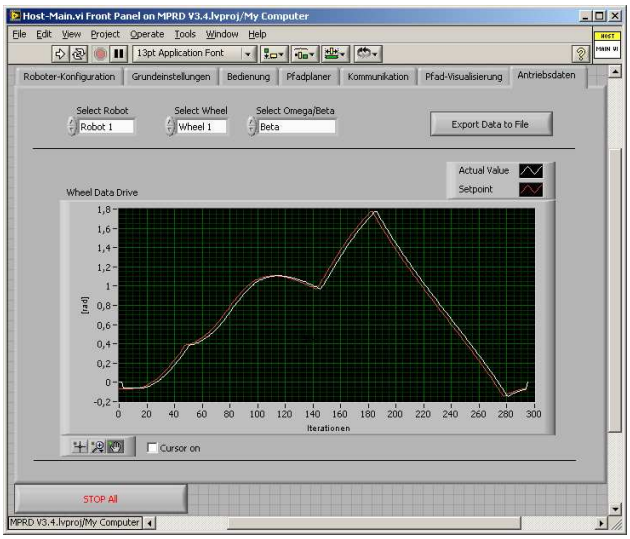


Abbildung 6.15: „Host-Main.vi“ - Registerkarte „Display Drive Data“

6.2.3 Programm „Master-MainDriveComb.vi“

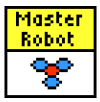


Abbildung 6.16: Icon des Master-MainDriveComb.vi

Das Programm „Master-MainDriveComb.vi“ wird auf der Recheneinheit (cRIO) des Masterroboterfahrwerkes ausgeführt. Es ist unter anderem für die Netzwerk-Verbindungsverwaltung, für die Weiterleitung der empfangenen Host-Kommandos an die Slaveroboterfahrwerke, für die Berechnung des Multiroboter-Bewegungsraumes $\mathbf{B}_m$ und für die Überprüfung der Ausführbarkeit der Multiroboter-Fahrbefehle $\dot{\boldsymbol{\xi}}_m$ zuständig. Nach dem Start des Programms „Master-MainDriveComb.vi“ werden alle Slave-Netzwerkteilnehmer (Host-PC und Slaveroboterfahrwerke) in die Netzwerkverbindungsliste aufgenommen. Bevor das Programm gestartet wird, muss das Robotermodell für das Roboterfahrwerk ausgewählt werden. Diese Auswahl wird im Front Panel des Programms, über das Bedienelement „Robotermodell“ getroffen. Das Robotermodell bzw. die Modelldefinition für ein Roboterfahrwerk muss im Vorfeld definiert werden. Nachdem das Robotermodell festgelegt wurde, kann das Programm „Master-MainDriveComb.vi“ gestartet werden. Am Beginn sendet das Programm roboterspezifische Daten wie Roboter-Name, IP-Adresse und Waben-Typ, die unter dem Meta-Element „CombInfo1“ zusammengefasst werden, an den Host-PC.

Diese Daten werden am Front Panel des „Host-Main.vi“ visualisiert. Nachdem die Daten des Meta-Elements „CombInfo1“ an den Host-PC gesendet wurden, werden die empfangenen Programmeinstellungen (Berechnungszykluszeit, Shiftparameter, Fahrmodus, ...) vom Host-PC im Programm „Master-MainDriveComb.vi“ gesetzt und im Front Panel visualisiert. Somit kann der Benutzer nochmals überprüfen, ob für das Masterroboterfahrwerk die gewünschten Grundeinstellungen vorliegen. In Abbildung 6.17 ist das Front Panel des „Master-MainDriveComb.vi“ dargestellt.
Nachdem die Programmeinstellungen gesetzt wurden, wartet das „Master-MainDriveComb.vi“ auf Kommandos, die am Front Panel des „Host-Main.vi“ (Registerkarte „Bedienung“) vom Benutzer in Auftrag gegeben werden. Es soll hier nochmal angemerkt werden, das bei der Verwendung der STM-Library keine Daten direkt von einem Slave-Netzwerkteilnehmer zu einem anderen Slave-Netzwerkteilnehmer übertragen werden können. Da der Host-PC ein Slave-Netzwerkteilnehmer ist, müssen die vom Host-PC an das Masterroboter-Fahrwerk gesendeten Kommandos, auch vom Masterroboter-Fahrwerk an die beteiligten Slaveroboterfahrwerke weitergeleitet werden. Im Folgenden soll kurz erläutert werden, welche Unterprogramme bzw. Aktionen am „Master-MainDriveComb.vi“ ausgeführt werden, wenn die angeführten Host-Kommandos vom Master-cRIO empfangen werden.

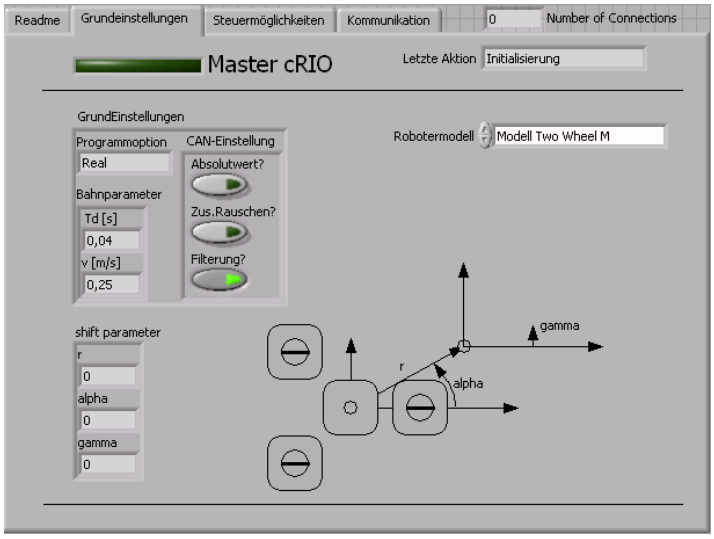


Abbildung 6.17: „Master-MainDriveComb.vi“ - Registerkarte „Grundeinstellungen“

Kommando - „Änderungen speichern“:
Die vom „Host-Main.vi“ gesendeten Programmeinstellungen werden in der Variablen „FGV Program Settings Host.vi“ gespeichert und im Anschluss im Programm

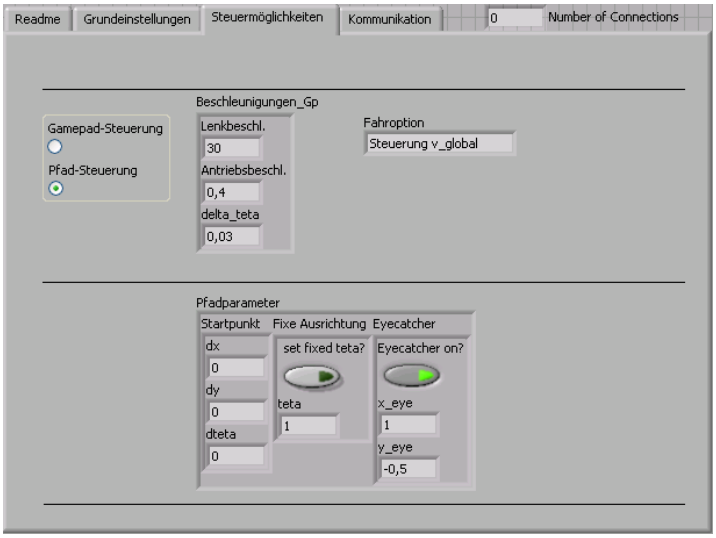


Abbildung 6.18: „Master-MainDriveComb.vi“ - Registerkarte „Steuermöglichkeiten“

„Master-MainDriveComb.vi“ gesetzt.

Kommando - „Init cRIO“:
Bei Empfang des Kommandos „Init cRIO“ am Master-cRIO wird zuerst das Programm „Init Robot.vi“ ausgeführt. Über dieses Programm werden die Motorregler „ELMO-Whistles“ des Masterroboter-Fahrwerkes parametriert und auf die Systemuhr des Master-cRIO sychronisiert. Zusätzlich wird noch der Bewegungsraum $\mathbf{B}_i$ für das Masterroboterfahrwerk berechnet.

Kommando - „Räder referenzieren“:
Beim Kommando „Räder referenzieren“ führt der Master-cRIO das Programm „Reference Robot.vi“ aus. Diesem Programm werden die Radlenkoffsetwinkel der einzelnen Radwaben, die in der Modelldefinition des zugehörigen Roboterfahrwerkes gespeichert sind, übergeben. Nach Ausführung des Programms „Reference Robot.vi“ zeigen alle Radachsen des Masterroboter-Fahrwerkes in Richtung Robotermittelpunkt $\mathbf{p}_{0,i}$ (siehe 4.1.9).

Kommando - „Roboter ausrichten“:
Wenn das Masterroboter-Fahrwerk das Kommando „Roboter ausrichten“ vom Host-PC empfängt, dann wir es in ein zeitgesteuertes Kommando umgewandelt. Das heißt, diesem Kommando wird vom Masterroboter-Fahrwerk ein absoluter Startzeitpunkt zugewiesen, bevor es an die Slaveroboterfahrwerke weitergeleitet wird. Der Startzeitpunkt wird

dabei so gewählt, dass alle Slaveroboterfahrwerke das Kommando „Roboter ausrichten“ garantiert empfangen haben und bereit für die Ausführung dieses Kommandos sind. Durch die synchronisierten Systemuhren der Roboterfahrwerke ist es möglich, trotz unterschiedlich langer Netzwerk-Kommunikationswege zwischen dem Host-PC und den Roboterfahrwerken, einen synchronen Start beim Ausrichten aller Roboterfahrwerke zu erreichen. Die Sequenz von Fahrbefehlen $[\dot{\boldsymbol{\xi}}_{m,1},\dot{\boldsymbol{\xi}}_{m,2},..,\dot{\boldsymbol{\xi}}_{m,k}]$, die für das Ausrichten des Roboterfahrwerkes notwendig ist, wurde in der Variablen „FGV Path Adjustment cRIO.vi“ am Master-cRIO gespeichert. Die Fahrbefehle $\dot{\boldsymbol{\xi}}_m$ werden vom Programm „PathGenerator.vi“ am „Host-Main.vi“ berechnet. Neben der Umwandlung des Kommandos „Roboter ausrichten“ in ein zeitgesteuertes Kommando, berechnet das Masterroboter-Fahrwerk bei Erhalt dieses Kommandos noch den Bewegungsraum $\mathbf{B}_m$ für das Multiroboterfahrwerk. Der Bewegungsraum $\mathbf{B}_m$ wird aufgrund der Bewegungsräume $\mathbf{Z}_i^*$ und $\bar{\mathbf{S}}_i^*$ aller beteiligten Roboterfahrwerke, über das Programm „calc Multispace B.vi“ berechnet.

Kommando - „Fahrt starten“:
Beim Empfang des Kommandos „Fahrt starten“ erfolgt ebenfalls eine Umwandlung in ein zeitgesteuertes Kommando, wie es beim Kommando „Roboter ausrichten“ beschrieben wurde. Der Unterschied zum Kommando „Roboter ausrichten“ liegt darin, dass beim Kommando „Fahrt starten“ kein Multiroboter-Bewegungsraum berechnet wird und das die Fahrbefehle für die Abfahrt des Sollpfades der Variablen „FGV Path“ entnommen werden.

Kommando - „Load/Refresh Drive Data“:
Bei Empfang dieses Kommandos sendet das Masterroboter-Fahrwerk die Antriebs- und Positionsdaten des Multiroboterfahrwerkes, die am Ende der Pfadabfahrt in der Variablen „Motion Data for Host.vi“ gespeichert wurden, an den Host-PC. In der Registerkarte „Antriebsdaten“ des „Host-Main.vi“ werden die Antriebs- und Positionsdaten des Multiroboterfahrwerkes grafisch dargestellt.

Kommando - „Stop All“:
Jede Bewegung des Masterroboter-Fahrwerkes wird gestoppt, die Netzwerkverbindungen zu den anderen Slave-Teilnehmern werden getrennt und das Programm „Master-MainDriveComb.vi“ wird beendet.

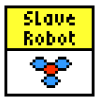


Abbildung 6.19: Icon des Slave-MainDriveComb.vi

6.2.4 Programm „Slave-MainDriveComb.vi“

Nach dem die Netzwerkverbindung vom jeweiligen Slave-cRIO (siehe Punkt 6.2.1) zum Master-cRIO aufgebaut wurde werden die festgelegten Programmeinstellungen am Front Panel des „Host-Main.vi“ auch im Programm „Slave-MainDriveComb.vi“ gesetzt. Das Programm „Slave-MainDriveComb.vi“ ist so aufgebaut, dass beim Programmablauf zwei Bewegungsszenarien des Slaveroboterfahrwerkes unterschieden werden können. Beim ersten Bewegungsszenario ist das Slaveroboterfahrwerk mit dem Masterroboter-Fahrwerk mechanisch verbunden oder das Slaveroboterfahrwerk bewegt sich synchron zum Masterroboter-Fahrwerk. In diesem Fall sendet das Programm „Slave-MainDriveComb.vi“ zu Ausführungsbeginn, die berechneten Slaveroboter-Bewegungsräume $\mathbf{Z}_i$ und $\bar{\mathbf{S}}_i$ an das Masterroboter-Fahrwerk. In weiterer Folge wartet das Programm „Slave-MainDriveComb.vi“ dann auf die eingehenden Multiroboterfahrbefehle $\dot{\boldsymbol{\xi}}_m$, die vom Masterroboter-Fahrwerk während der Fahrt des Multiroboter-Fahrwerkes an die Slaveroboterfahrwerke gesendet werden. Diese Multiroboter-Fahrbefehle $\dot{\boldsymbol{\xi}}_m$ wurden vom Masterroboter-Fahrwerk schon auf ihre Ausführbarkeit hin überprüft und müssen vom Programm „Slave-MainDriveComb.vi”nur mehr in den lokalen Slaveroboter-Mittelpunkt $\mathbf{p}_{0,i}$ transformiert werden. Diese transformierten Fahrbefehle $\dot{\boldsymbol{\xi}}_i$ werden der Inverskinematik übergeben, die die Radlenkwinkel und Raddrehzahlen für die einzelnen Slaveroboterfahrwerksräder berechnet. Beim zweiten Bewegungsszenario bewegt sich das Slaveroboterfahrwerk variabel also unabhängig zum Masterroboter-Fahrwerk. In diesem Fall muss für das Slaveroboterfahrwerk ein eigener Sollpfad am „Host-Main.vi“ vorgegeben werden. Durch die Unabhängigkeit zum Masterroboter-Fahrwerk muss das Programm „Slave-MainDriveComb.vi“ jetzt auch die Fahrbefehlüberprüfung selbst durchführen. Das Programm „Slave-MainDriveComb.vi“ ist in der Multiroboter-Softwareversion „MPRD V3.4“ schon so ausgelegt worden, dass es auf die vorhin beschriebenen zwei Bewegungsszenarien des Slaveroboterfahrwerkes reagieren kann. Bei den Programmen „Host-Main.vi“ und „Master-MainDriveComb.vi“ ist das noch nicht der Fall. Von der grafischen Programmoberfläche (Front Panel) ist das Programm „Slave-MainDriveComb.vi“ mit dem Programm „Master-MainDriveComb.vi“ nahezu ident. In Abbildung 6.20 und

6.21 ist das Front Panel des „Slave-MainDriveComb.vi“ dargestellt. Beim Programm „Slave-MainDriveComb.vi“ ist der Registerkarte „Steuermöglichkeiten“ noch zusätzlich ein Anzeigeelement für die „Master-Steuerung“ eingefügt worden. Damit wird angezeigt, dass das Slaveroboterfahrwerk vom Masterroboterfahrwerk gesteuert wird.

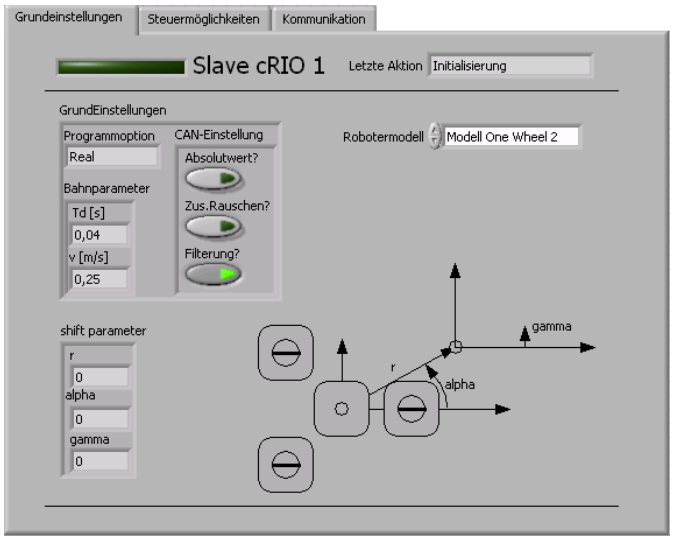


Abbildung 6.20: „Slave-MainDriveComb.vi Registerkarte „Grundeinstellungen“

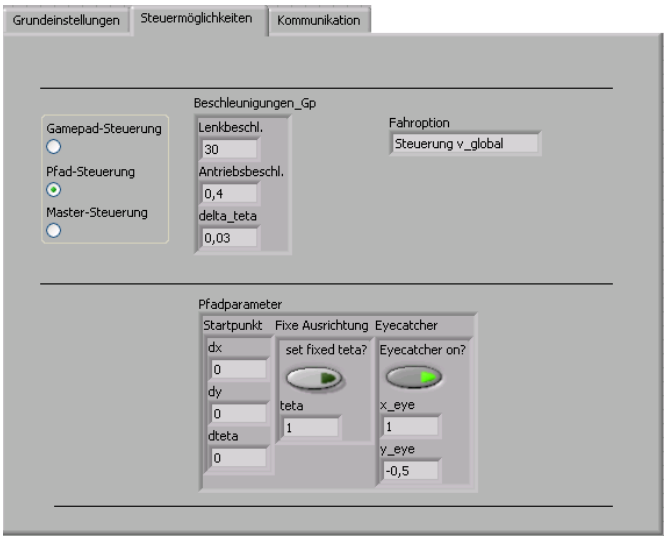


Abbildung 6.21: „Slave-MainDriveComb.vi Registerkarte „Steuermöglichkeiten“

Im Folgenden soll kurz erläutert werden, welche Unterprogramme bzw. Aktionen am „Slave-MainDriveComb.vi“ ausgeführt werden, wenn die angeführten Host-Kommandos bzw. die vom Masterroboterfahrwerk weitergeleiteten Host-Kommandos am Slave-cRIO

empfangen werden.

Kommando - „Änderungen speichern“:
Die vom Master-cRIO weitergeleiteten Host-Programmeinstellungen werden in der Variablen „FGV Program Settings Host.vi“ gespeichert und im Anschluss im Programm „Slave-MainDriveComb.vi“ gesetzt.

Kommando - „Init cRIO“:
Bei Empfang des Kommandos „Init cRIO“ am Slave-cRIO wird zuerst das Programm „Init Robot.vi“ ausgeführt. Über dieses Programm werden die Motorregelgeräte (ELMO-Whistles) des Slaveroboterfahrwerkes parametriert und auf die Systemuhr des Slave-cRIO synchronisiert. Zusätzlich wird noch der Bewegungsraum $\mathbf{B}_i$ für das Slaveroboterfahrwerk berechnet.

Kommando - „Räder referentieren“:
Beim Kommando „Räder referenzieren“ führt der Slave-cRIO das Programm „Reference Robot.vi“ aus. Diesem Programm werden die Radlenkoffsetwinkel der einzelnen Radwaben, die in der Modelldefinition des ausgewählten Slaveroboterfahrwerkes gespeichert sind, übergeben. Nach Ausführung des Programms „Reference Robot.vi“ zeigen alle Radachsen des Slaveroboterfahrwerkes in Richtung Robotermittelpunkt $\mathbf{p}_{0,i}$.

Kommando - „Roboter ausrichten“:
Wenn der Slave-cRIO das Kommando „Roboter ausrichten“ empfängt, dann wurde es schon vom Masterroboterfahrwerk in ein zeitgesteuertes Kommando umgewandelt. Das heißt, das Slaveroboterfahrwerk braucht nur mehr zu warten bis der Zeitpunkt eintritt, an dem das zeitgesteuerte Kommando „Roboter ausrichten“ seine Gültigkeit erlangt, und im Anschluss die empfangen Fahrbefehle vom Masterroboterfahrwerk ausführen.

Kommando - „Fahrt starten“:
Das Kommando „Fahrt starten“ ist für das Slaveroboterfahrwerk ident zum Kommando „Roboter ausrichten“. Das Unterschied liegt darin, dass vom Masterroboterfahrwerk Fahrbefehle $\dot{\boldsymbol{\xi}}_m$ gesendet werden, die sich auf den abzufahrenden Sollpfad beziehen.

Kommando - „Stop All“:
Jede Bewegung des Slaveroboterfahrwerkes wird gestoppt und das Programm „Slave-MainDriveComb.vi“ wird beendet.

6.2.5 Low-Level-Softwaremodul

Das Low-Level-Softwaremodul ist hierarchisch gesehen das niederwertigste Software-Modul unter den Multiroboter-Programmen und wird auf jeder Recheneinheit (cRIO) eines Roboterfahrwerkes ausgeführt. Somit wird das Low-Level-Softwaremodul auch von den Programmen „Master-MainDriveComb.vi“ und „Slave-MainDriveComb.vi“ aufgerufen. Das Low-Level-Softwaremodul ist unter anderem für die Parametrierung der Motorregelgeräte (ELMO-Whistles) der Radwaben, für den Datentransfer zwischen Low-Level-Network und den Motorregelgeräten (ELMO-Whistles), für den Start der Position-Time-Tables (PTTs) auf den ELMO-Whistles und für die Synchronisation von Recheneiheit (cRIO) und den Motorregelgeräten (ELMO-Whistles) verantwortlich. Das Low-Level-Network ist dabei als definierte Daten-Schnittstelle zu übergeordneten Programmteilen (High-Level-Softwaremodule) aufzufassen.

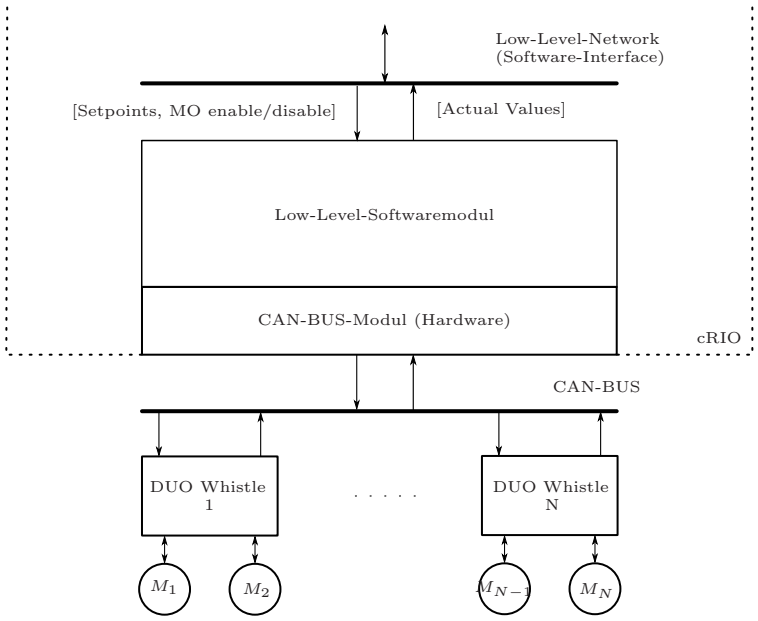


Abbildung 6.22: Lage des Low-Level-Softwaremoduls

6.2.5.1 Definition der Modul-Schnittstelle (Low-Level-Network)

In Abbildung 6.23 ist die Datenschnittstelle des Low-Level-Softwaremoduls grafisch dargestellt. Über diese werden Daten wie Motorregelgeräte-IDs (Node-IDs), Sollwertvorgaben für die Radwaben (Setpoints), Istwerte von Radwaben (Actual Values) und Steuersignale (Motion enable/disable) zur Aktivierung und Deaktivierung der Radwaben-

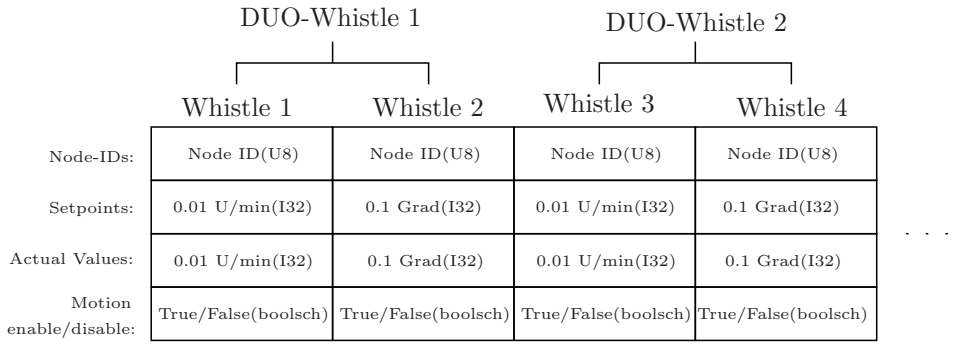


Abbildung 6.23: Low-Level-Softwaremodul - Schnittstellendefinition

Antriebsmotoren übertragen. Die Größe der Daten-Arrays (Node-IDs, Setpoints,...) ist von der Anzahl der Radwaben und somit von der Anzahl der ELMO-DUO-Whistle-Boards, die von einer Recheneinheit (cRIO) versorgt werden, abhängig. Pro Radwabe bzw. ELMO-DUO-Whistle-Board kommen 2 Elemente zu den Daten-Arrays hinzu.

6.2.5.2 Beschreibung der wichtigsten Programmteile

Wie schon eingangs erwähnt wurde, sind die Hauptaufgaben des Low-Level-Softwaremodules die Parametrierung der Motorregelgeräte (ELMO-Whistles), der Datenaustausch zwischen Low-Level-Netzwerk und den Motorregelgeräten (ELMO-Whistles) und die Synchronisation zwischen der Recheneinheit (cRIO) und den Motorregelgeräte (ELMO-Whistles). In diesem Abschnitt werden die wichtigsten LabVIEW-VIs vorgestellt, die für die Funktion der vorhin beschriebenen Aufgaben verantwortlich sind. Es soll auch ein Überblick über die hierarchische Lage der beschriebenen LabVIEW-VIs geschaffen werden.

6.2.5.3 Initialisierung/Parametrierung

In der High-Level-Sofwareebene („Master-MainDriveComb.vi“ und „Slave-MainDriveComb.vi“) wird zur Initialisierung der ELMO-Whistles das Programm „CAN Init.vi“ aufgerufen. Diesem Programm werden die Node-IDs der zu initialisierenden Motorregler (ELMO-Whistles) übergeben. Im Programm „CAN Mapping.vi“ werden die im Abschnitt A.1.4 angeführten Einstellungen gesetzt. Damit sind die Motorregelgeräte (ELMO-Whistles) so konfiguriert, dass sie bei Erhalt eines CAN-SYNC-Kommandos, die aktuellen Istwerte (Raddrehzahl und Radlenkwinkel) der Radwaben, über den CAN-Bus an die Recheneinheit (cRIO) des Roboterfahrwerkes

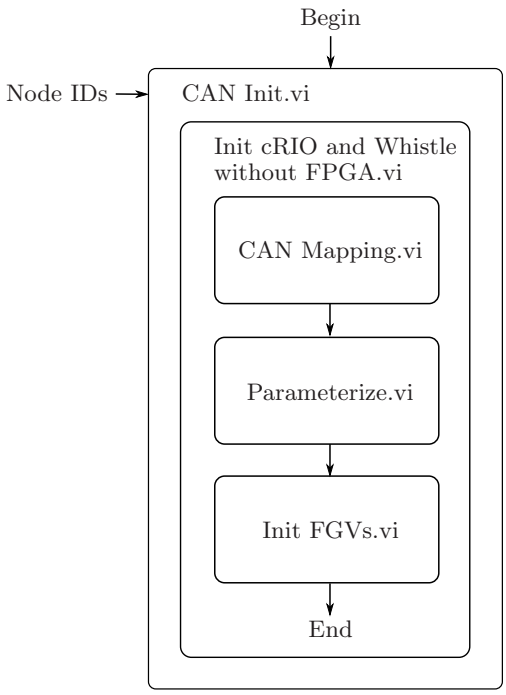


Abbildung 6.24: VI-Hierarchie „CAN Init.vi“

senden. Das Programm „Parameterize.vi“ setzt unter anderem die notwendigen Einstellungen für die Position-Time-Tables (PTTs), die im Abschnitt A.1.5 angeführt sind. Zusätzlich werden die μs-Systemuhren der Motoregler (ELMO-Whistles) auf die der Recheneinheit (cRIO) gestellt. Eine genaue Synchronisation erfolgt im Anschluss mittels 50 gesendeten SYNC-Timestamp-Paaren (siehe Punkt 5.1.1). Im Programm „Init FGVs.vi“ werden alle notwendigen „Funktionalen Globalen Variablen“ (FGVs), die im Low-Level-Softwaremodul verwendet werden, initialisiert. Nach erfolgreicher Initialisierung aller Hardwarekomponenten und Softwaremodule kann der Datenaustausch zwischen Low-Level-Softwaremodul und den übergeordneten Programmteilen (High-Level-Softwaremodule) erfolgen.

6.2.5.4 Sollwertvorgabe über Position-Time-Table (PTT)

Nachdem die Initialisierung aller Hardwarekomponenten und Softwaremodulen abgeschlossen wurde und die Radlenkeinheiten der Radwaben referenziert wurden, ist das Roboterfahrwerk bereit einen vorgegebenen Pfad abzufahren. Dabei müssen die berechneten Radlenkwinkel [0.1°] und Raddrehzahlen [0.01 U/min] für die Position-Time-Table (PTT) über das Programm „Setpoint2rawdata.vi“ in Inkremente pro Abtastschritt [Ink./T_d] um-

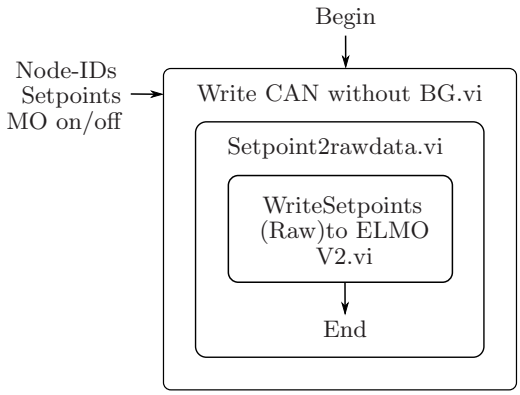


Abbildung 6.25: VI-Hierarchie „Write CAN without BG.vi“

gerechnet werden. In den Gleichungen 6.1 und 6.2 ist die Umrechnung der Sollwertvorgaben, von Radlenkwinkel und Raddrehzahl, in Inkremente pro Abtastschritt angegeben. Die berechneten Positionswerte pro Abtastschritt [Ink./T_d] werden über das SimpleIQ-Kommando [5] „QP[Index PTT] = Absoluter Positionswert(I32)“ per CAN-Bus in die entsprechenden Speicherstellen der Position-Time-Table (PTT) geschrieben.

$$Raddrehzahl \left[\frac{0.01\,U}{min}\right] \cdot \frac{T_d\,[ms] \cdot 182}{36000} + \left[\frac{Ink.}{T_d}\right]_{k-1} = \left[\frac{Ink.}{T_d}\right]_k \tag{6.1}$$

$$Lenkwinkel\,[0.1^\circ] \cdot \frac{48148}{1000} = \left[\frac{Ink.}{T_d}\right]_k \tag{6.2}$$

Eine volle Umdrehung der Radlenkeinheit ist mit 173777.3$\dot{}$ Inkrementen aufgelöst (siehe Projektarbeit [8]), wodurch sich eine Lenkwinkeländerung von einem Zehntel-Grad in einer Drehencoder-Werteänderung von 48.148 Inkrementen niederschlägt.

6.2.5.5 Einlesen der aktuellen Radlenkwinkel und Raddrehzahlwerte

Wie im Abschnitt A.1.4 beschrieben wurde, wird ein Motorregelgerät (ELMO-Whistle) so konfiguriert, dass es bei Erhalt eines CAN-SYNC-Kommandos, den aktuellen Positionswert des Drehencoders per CAN-Bus an die Recheneinheit des Roboterfahrwerkes (cRIO) senden. Über das Programm „ReadActualValues(Raw)fromELMO V2.vi“ werden die Positionswerte der Drehencoder, die auf der Recheneinheit (cRIO) gespeichert werden, ausgelesen und dem Low-Level-Softwaremodul zur Verfügung gestellt. Diese Positions-Rohdaten [Ink.] werden vom Programm „Rawdata2ActualValues“ wieder in die entsprechenden Lenkwinkel [0.1°] oder Raddrehzahlen [0.01 U/min] umgerechnet. In den Glei-

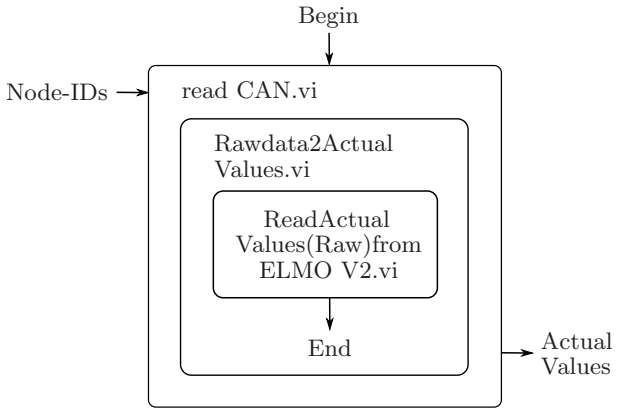


Abbildung 6.26: VI-Hierarchie „read CAN.vi“

chungen 6.3 und 6.4 ist die Umrechnung der Positions-Rohdaten in Inkremente angegeben.

$$Pos._{(k)}\,[Inkr.] \cdot \frac{1000}{48148} = Lenkwinkel\,[0.1^{\circ}] \tag{6.3}$$

$$Pos._{(k)}\,[Inkr.] - Pos._{(k-1)}\,[Inkr.] \cdot \frac{36000}{182 \cdot T_d[ms]} = Drehzahl\,\left[\frac{0.01U}{min}\right] \tag{6.4}$$

$Pos._{(k)}$ [Inkr.] ... Der aktuelle Positionswert des Drehencoders in Inkremente.
$Pos._{(k-1)}$ [Inkr.] ... Der letzte Positionswert des Drehencoders in Inkremente.

Kapitel 7

Testfahrten

Im diesem Kapitel werden die Ergebnisse aus drei unterschiedlichen Testfahrten vorgestellt. Dabei wurde versucht, dass jede Testfahrt die praktische Umsetzung eines anderen Themas beleuchtet, das zuvor in dieser Arbeit theoretisch ausgearbeitet wurde. Die Testfahrt 1 soll zeigen, dass die Roboterhardware bzw. die Multirobotersoftware auch in der Lage ist, Roboterfahrwerke mit mehr als 3 Radwaben zu bedienen. Bei der Testfahrt 2 wird gezeigt, dass die in der Multirobotersoftware „MPRD V3.4“ umgesetzten Berechnungsvorschriften, die aus der theoretischen Ausarbeitung des Multiroboterszenarios gewonnen wurden, sich auch auf ein reales Multiroboterfahrwerk übertragen lassen. Die Testfahrt 3 setzt sich abschließend mit der praktischen Umsetzung des Multiroboterszenarios „Vollständig verteilter Fall“ auseinander, bei dem jede Radwabe mit einer eigenen Recheneinheit (cRIO) ausgestattet ist. Die für die Testfahrten verwendeten Roboterfahrwerke sind alle mit Radwaben aufgebaut, die mit Standardrädern bestückt sind. Die Standardräder können dabei gelenkt und angetrieben werden.

7.1 Testfahrt 1 - Roboterfahrwerk mit 5 Radwaben

In den bisher durchgeführten Testfahrten [13] wurden Roboterfahrwerke bestehend aus maximal 3 Radwaben aufgebaut. Mit dieser Testfahrt soll gezeigt werden, dass mittels der entworfenen Multirobotersoftware „MPRD V3.4“ und der verwendeten Roboterhardware (siehe Kapitel 2) auch Roboterfahrwerke mit mehr als 3 Radwaben pro Recheneinheit (cRIO) bedient werden können. Im vorliegenden Fall wurde ein Roboterfahrwerk bestehend aus 5 Radwaben aufgebaut. In Abbildung 7.1 ist die Konfiguration des 5-rädrigen Roboterfahrwerkes mit dem festgelegten Robotermittelpunkt $\mathbf{p}_{0,1}$ und dem lokalen Robo-

terkoordinatensystem $\Sigma_{R,1}$ grafisch dargestellt.

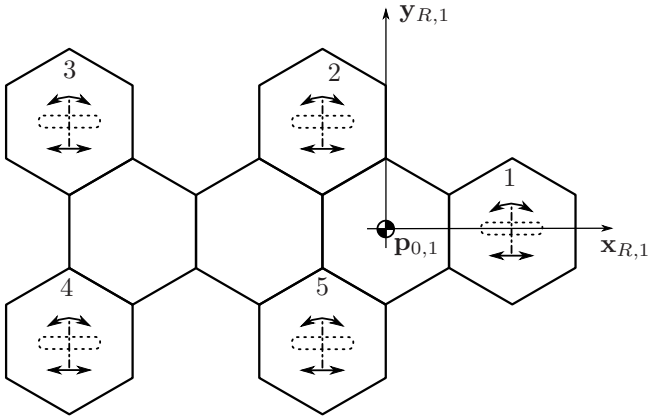


Abbildung 7.1: 5-rädriges Roboterfahrwerk - Testfahrt 1

Der festgelegte abzufahrende Sollpfad für das Roboterfahrwerk, ist in Abbildung 7.2 dargestellt. Das rote Kreuz legt dabei den Startpunkt des Roboterfahrwerkes fest und der rote Pfeil zeigt die Richtung an, in der das Roboterfahrwerk den vorgegeben Sollpfad abfährt. Dieser Pfad wird vom 5-rädrigen Roboterfahrwerk im Fahrmodus „Fixed Theta“ abgefahren. Dabei behält der Roboterfahrwerk während der Fahrt einen konstanten Winkel von $\Theta = 0$ rad zum raumfesten globalen Roboterkoordinatensystem Σ_I bei. Die Ausrichtung des Roboterfahrwerkes während der Fahrt wird durch die blauen Pfeile (Abb. 7.2) gekennzeichnet. Der aufgrund der Odometriedaten (Abb. 7.4 und 7.5) errechnete Istpfad, wird in

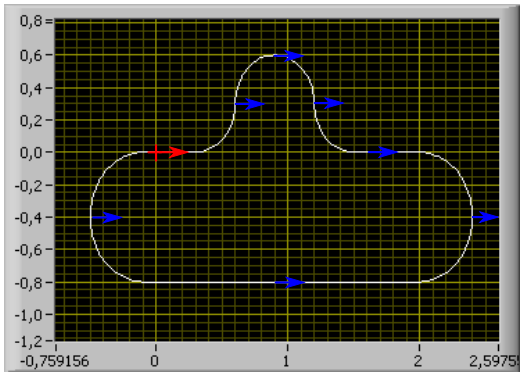


Abbildung 7.2: Vorgegebener Sollpfad - Testfahrt 1

Abbildung 7.3 mit dem vorgegeben Sollpfad verglichen. Es ist zu erkennen, dass die Abweichung zwischen Soll- und Istpfad besonders in Kurvenfahrten auftritt. Der Grund für diese Abweichung ist folgender. Die vom Pfadplaner vorgegeben globalen Fahrbefehle $\dot{\boldsymbol{\xi}}_I$ müssen über die Transformationsvorschrift 4.4 in das lokale Roboterkoordiantensystem $\Sigma_{R,i}$ umge-

rechnet werden. Der für die Rotationsmatrix 4.5 benötigte Winkel Θ (Winkel zwischen x_I- und $x_{R,i}$-Achse) wird bei der aktuellen Programmversion aus den Odometriedaten gewonnen. Aufgrund der Interpolation (3. Ordnung) zwischen den Sollwertvorgaben, ergibt sich eine Verzögerung von 3 Abtastschritten, zwischen der Vorgabe und der Ausführung eines lokalen Roboterfahrbefehls $\dot{\boldsymbol{\xi}}_{R,i}$. Damit ist auch der ermittelte Winkel Θ aus den Odometriedaten „verzögert“, wodurch sich ein Fehler bei der Umrechnung vom globalen $\dot{\boldsymbol{\xi}}_I$ in den lokalen Geschwindigkeitsvektor $\dot{\boldsymbol{\xi}}_R$ ergibt. Darin liegt der Grund für die Abweichung zwischen Soll- und Istpfad. In Abbildung 7.4 und 7.5 sind die Radlenkwinkel- und Rad-

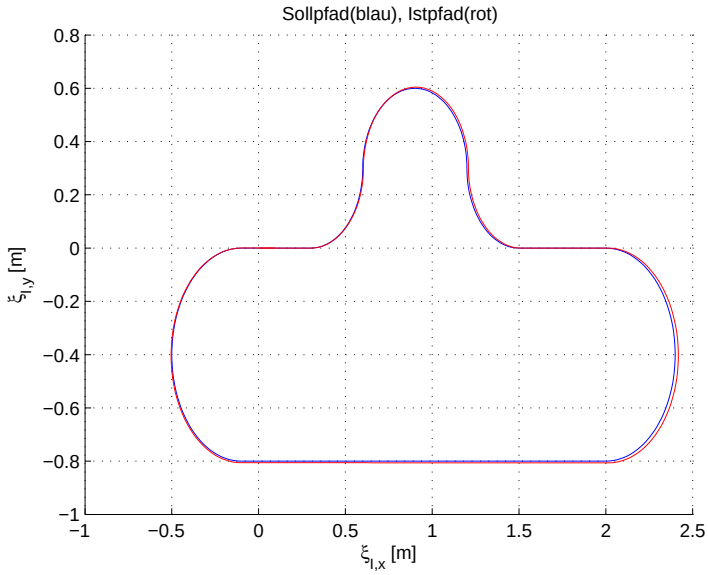


Abbildung 7.3: Vergleich von Soll- und Istpfad - Testfahrt 1

drehzahldaten $\beta_{k_i,i}$ und $\omega_{k_i,i}$ der einzelnen Roboterfahrwerksräder grafisch dargestellt, die während der Fahrt des Roboterfahrwerkes aufgezeichnet wurden. Werden die Diagramme der Raddrehzahlen genauer betrachtet, dann fällt auf, dass in den ersten 0.8 Sekunden die Raddrehzahl der einzelnen Roboterfahrwerksräder bei Null liegt. Das kann damit erklärt werden, dass das Roboterfahrwerk mit der Abfahrt solange wartet, bis jede Radwabe ihre erste Radlenkwinkel-Sollvorgabe angefahren hat. Erst dann beginnt das Roboterfahrwerk den vorgegeben Sollpfad abzufahren.

Fazit - Testfahrt 1:
Mit dieser Testfahrt konnte gezeigt werden, dass mit der vorliegenden Roboterhardware und dem Multiroboterprogramm „MPRD V3.4“ auch Roboterfahrwerke mit mehr als 3 Radwaben pro Recheneinheit problemlos bedient werden können. Es muss jedoch darauf

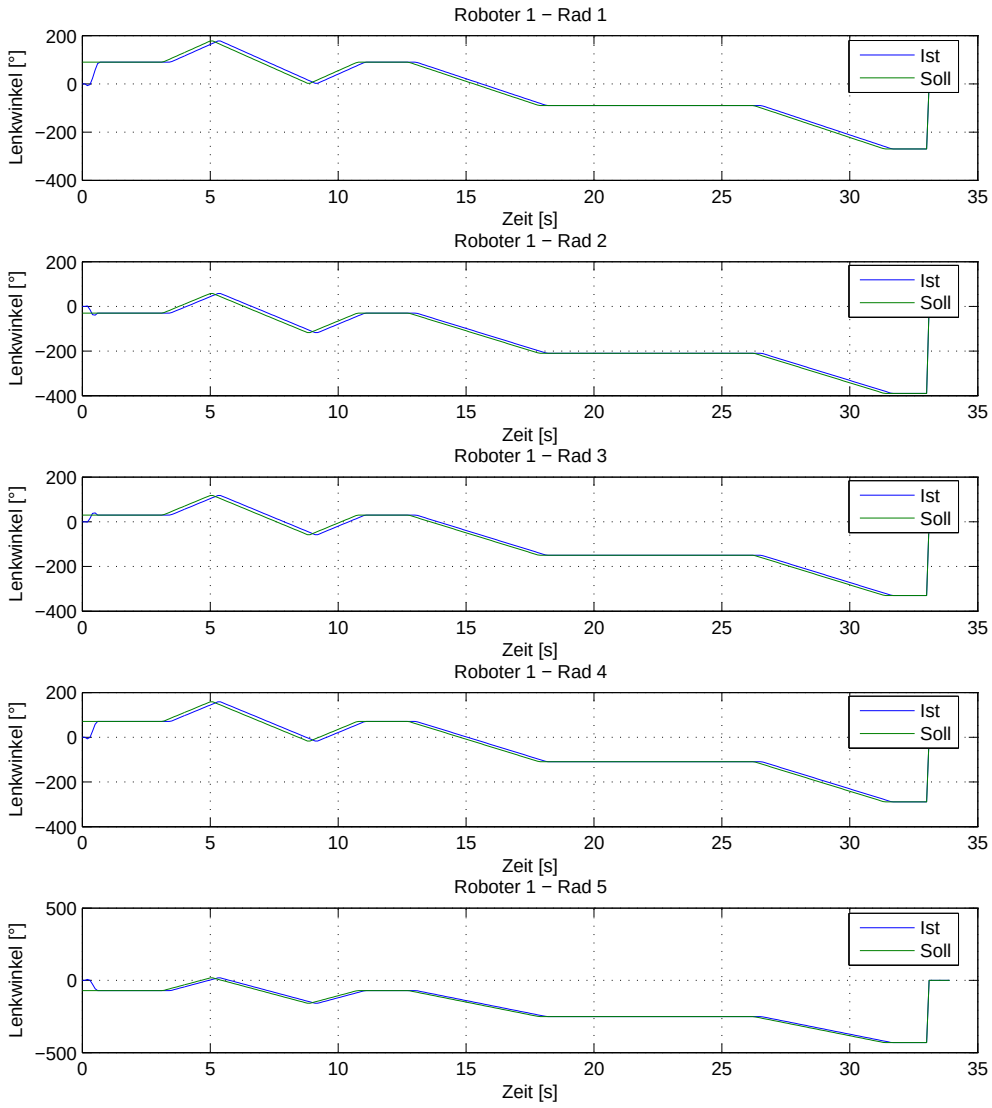


Abbildung 7.4: Radlenkwinkeldaten - Testfahrt 1

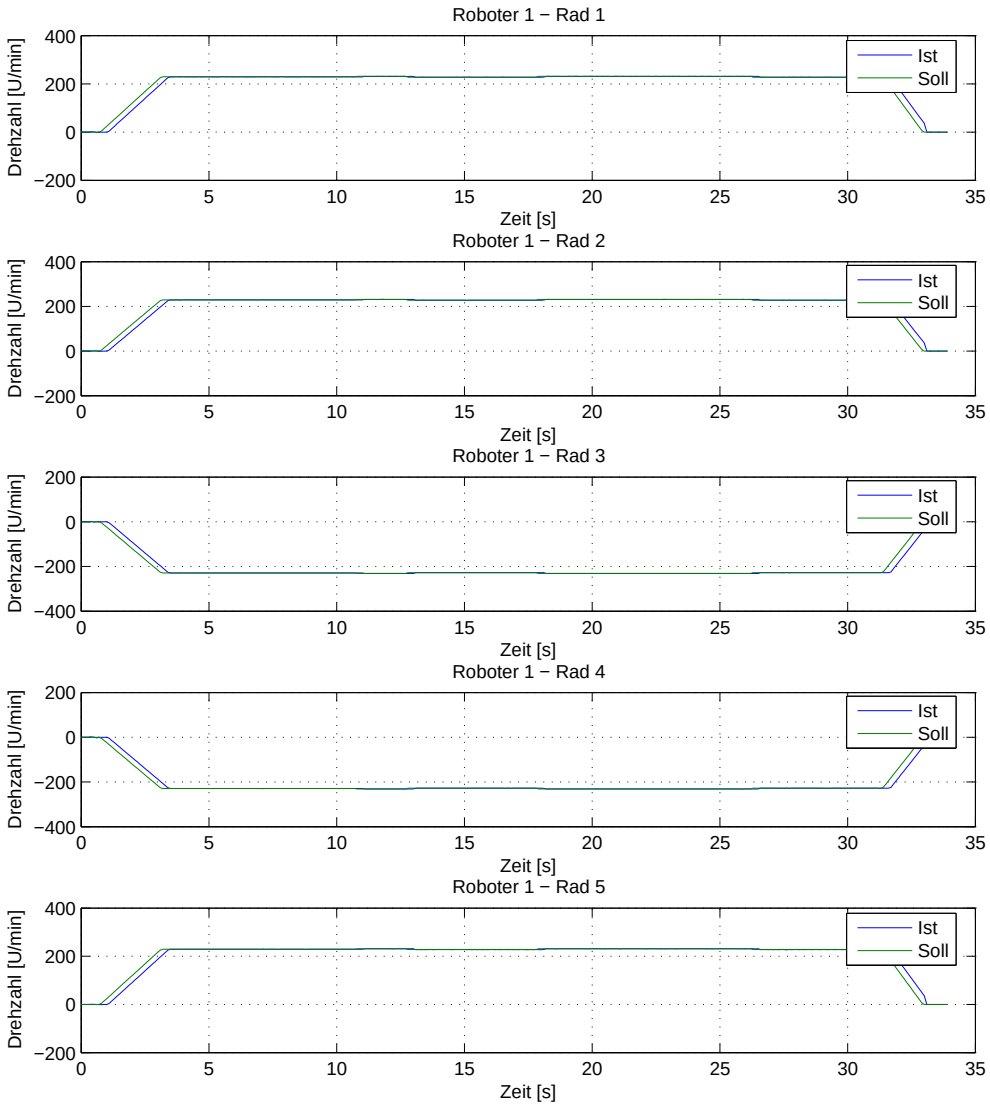


Abbildung 7.5: Raddrehzahldaten - Testfahrt 1

geachtet werden, dass die benötigte Zeit für das Einlesen der aktuellen Radlenkwinkel- und Raddrehzahlwerte und das Setzen neuer Radlenkwinkel- und Raddrehzahlwerte mit der Anzahl der Radwaben steigt. Deshalb muss bei mehr als 6 Radwaben pro Recheneinheit das Low-Level-Softwaremodul entsprechend adaptiert werden, da es aktuell für eine maximale Radwabenanzahl von 6 Stück pro Recheneinheit ausgelegt ist.

7.2 Testfahrt 2 - Multiroboterfahrwerk

Mit dieser Testfahrt soll gezeigt werden, dass die in der Multirobotersoftware „MPRD V3.4“ umgesetzten Berechnungsvorschriften, die aus der theoretischen Ausarbeitung des Multiroboterszenarios gewonnen wurden, sich auch auf ein reales Multiroboterfahrwerk übertragen lassen. Das für die Testfahrt verwendete Multiroboterfahrwerk besteht aus zwei Roboterfahrwerken, wobei das Masterroboterfahrwerk aus 3 Radwaben und Slaveroboterfahrwerk aus zwei Radwaben aufgebaut ist. In Abbildung 7.6 ist die Konfiguration des Masterroboterfahrwerkes mit dem lokalen Masterroboterfahrwerks-Mittelpunkt $\mathbf{p}_{0,1}$ und die Konfiguration des Slaveroboterfahrwerkes mit dem lokalen Slaveroboterfahrwerks-Mittelpunkt $\mathbf{p}_{0,2}$ dargestellt. Beide Roboterfahrwerke zusammen bilden das Multiroboterfahrwerk. Es soll hier angemerkt werden, das die beiden Roboterfahrwerke nicht mechanisch verbunden, sondern nur zusammen geschoben wurden. Trotzdem sollten sie während der Multiroboterfahrt ihre Position zueinander nicht verändern.

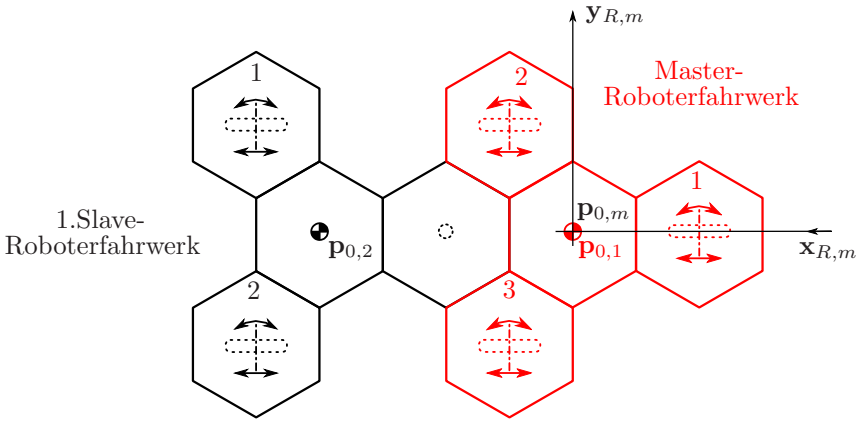


Abbildung 7.6: Multiroboterfahrwerk - Testfahrt 2

In Abbildung 7.7 ist der abzufahrende Sollpfad für das Multiroboterfahrwerk dargestellt. Es handelt sich dabei um den gleichen Sollpfad, der schon für das 5-rädrige Roboterfahrwerk in der Testfahrt 1 verwendet wurde. Das rote Kreuz legt dabei den Startpunkt des

Multiroboterfahrwerkes fest und der rote Pfeil zeigt die Richtung an, in der das Multiroboterfahrwerk den vorgegeben Sollpfad abfährt. Dieser Pfad wird vom Multiroboterfahrwerk im Fahrmodus „Fixed Theta“ abgefahren. Dabei behält das Multiroboterfahrwerk während der Fahrt einen konstanten Winkel von $\Theta = 0$ rad zum raumfesten globalen Roboterkoordinatensystem Σ_I bei. Die Ausrichtung des Mulriroboterfahrwerkes während der Fahrt, wird durch die blauen Pfeile im Pfaddiagramm (Abb. 7.7) dargestellt. Der auf-

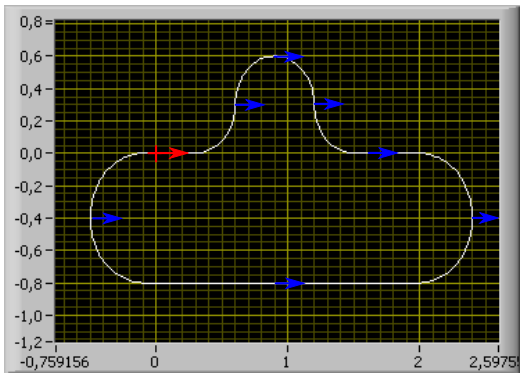


Abbildung 7.7: Vorgegebener Sollpfad - Tesfahrt 2

grund der Odometriedaten (Abb. 7.9 und 7.10) errechnete abgefahrene Istpfad, wird in Abbildung 7.8 mit dem vorgegeben Sollpfad verglichen. Es ist zu erkennen, dass die Positionsabweichung vom Sollpfad für beide Roboterfahrwerke nicht sehr groß ist. Was jedoch in diesem Positionsdiagramm nicht zu erkennen ist, ist die zeitliche Abweichung zwischen Master- und Slave-Istpfad, aufgrund von zu spät empfangenen Slaveroboter-Fahrbefehlen. In Abbildung 7.9 und 7.10 sind die Radlenkwinkel- und Raddrehzahldaten der einzelnen Roboterfahrwerksräder grafisch dargestellt, die während der Fahrt des Multiroboterfahrwerkes aufgezeichnet wurden. Der Grund warum die Raddrehzahl in Abbildung 7.10 in den ersten 0.8 Sekunden bei Null liegt ist, dass das Multiroboterfahrwerk solange mit der Abfahrt wartet, bis jede Radwabe ihre erste Radlenkwinkel-Sollvorgabe angefahren hat. Erst dann beginnt das Multiroboterfahrwerk mit der Pfadabfahrt. Werden die vorgegeben Radlenkwinkel und Raddrehzahlen von der Testfahrt 1 mit der Testfahrt 2 verglichen, dann fällt auf, dass trotz gleicher Sollpfadvorgabe unterschiedliche Radlenkwinkel und Raddrehzahlen vorgegeben werden. Der Grund liegt darin, dass sich die Lage der lokalen Robotermittelpunkte $\mathbf{p}_{0,i}$ bei den beiden Roboterfahrwerken unterscheidet und infolgedessen auch die Modelldefinitionen der Roboterfahrwerke unterschiedlich sind. Dadurch ändern sich auch die Radlenkwinkel- und Raddrehzahlvorgaben der Roboterfahrwerke, trotz gleicher Sollpfadvorgabe.

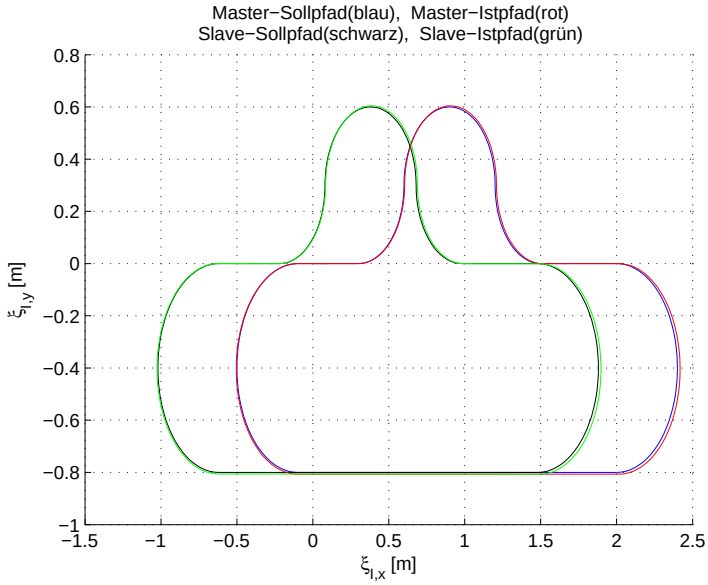


Abbildung 7.8: Vergleich von Soll- und Istpfad - Testfahrt 2

Fazit - Testfahrt 2:

Bei der Testfahrt 2 konnte während der Fahrt des Multiroboterfahrwerkes manchmal ein kleiner Spalt zwischen dem Slave- und dem Masterroboterfahrwerk beobachtet werden (Die Roboterfahrwerke wurden nicht mechanisch verbunden). Der Grund für diese Abweichung ist, dass das Slaveroboterfahrwerk die gesendeten Fahrbefehle $\dot{\boldsymbol{\xi}}_m$ vom Masterroboterfahrwerk zu spät empfängt. Dadurch kommt es zu den Effekten, dass das Masterroboterfahrwerk schon fährt, wenn das Slaveroboterfahrwerk noch steht oder das Slaveroboterfahrwerk lenkt später in die Kurve ein als das Masterroboterfahrwerk. Wenn das Kommunikationsproblem (siehe Punkt 5.2.4.3) zwischen dem Master-cRIO und den Slave-cRIOs gelöst wurde, dann sollten die Abweichungen des Slaveroboterfahrwerkes vom vorgegeben Sollpfad auch der Vergangenheit angehören. Trotz kleinerer Abweichung konnte gezeigt werden, dass das Multiroboterfahrwerk den vorgegeben Sollpfad korrekt abfährt. Damit wurde gezeigt, dass die theoretisch ausgearbeiteten Berechnungsvorschriften für das Multiroboterszenario auch mit der Multirobotersoftware „MPRD V3.4“ erfolgreich umgesetzt werden konnten.

7.3 Tesfahrt 3 - MRZ „Vollständig Verteilter Fall“

Diese Testfahrt zeigt die praktische Umsetzung der Multiroboterkonfiguration „Vollständig Verteilter Fall“. In diesem Fall besitzt jede Radwabe ihre eigene Recheneinheit (cRIO).

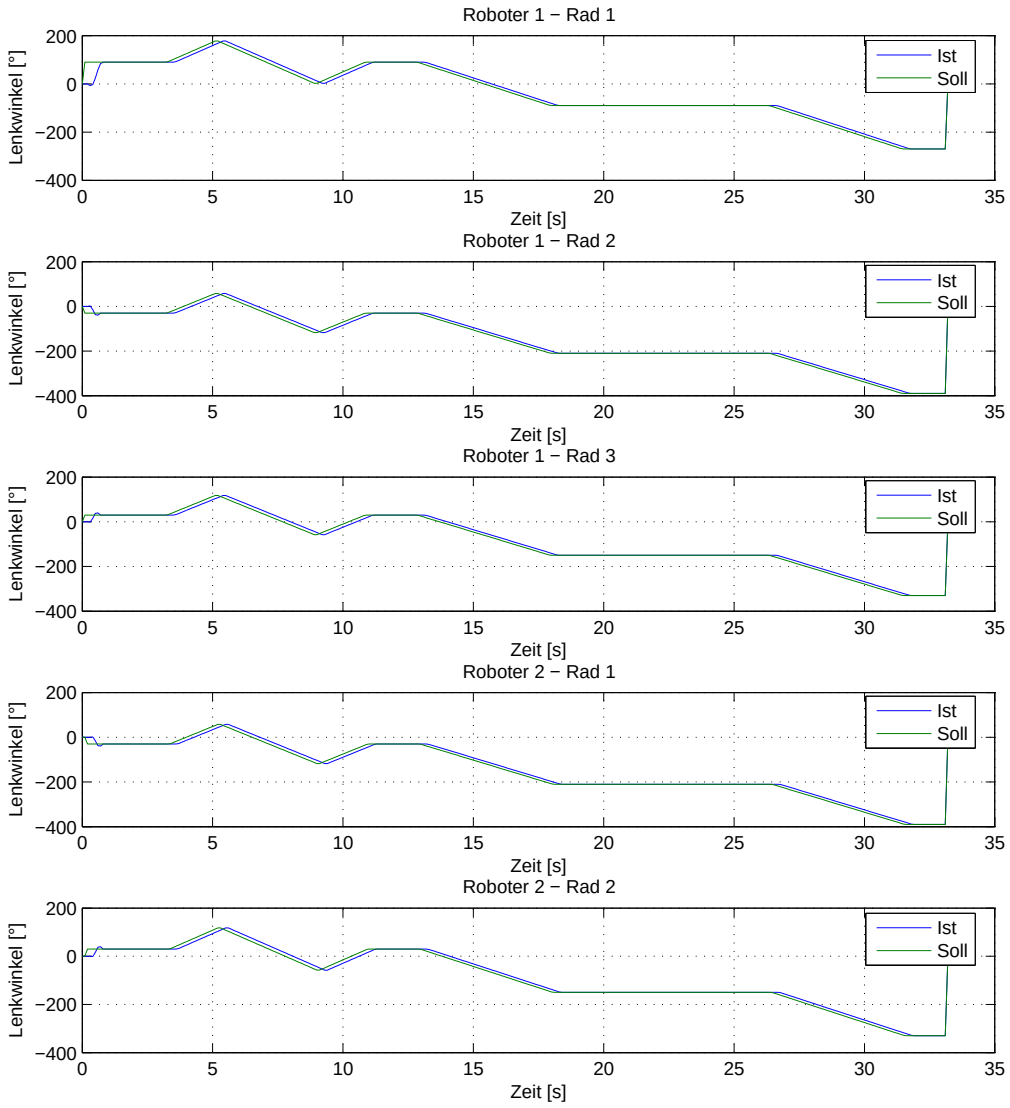


Abbildung 7.9: Radlenkwinkeldaten - Testfahrt 2

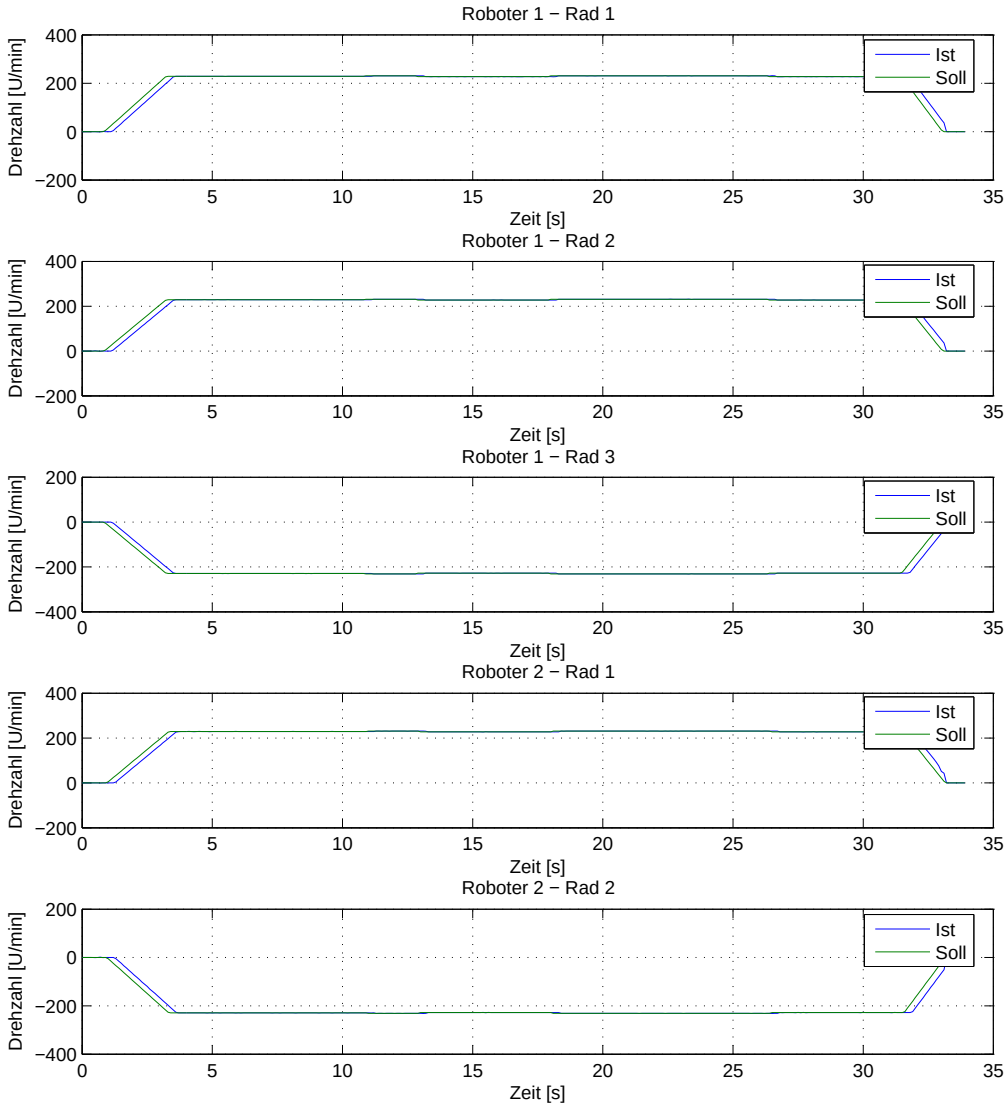


Abbildung 7.10: Raddrehzahldaten - Testfahrt 2

In Abbildung 7.11 ist die Konfiguration des Multiroboterfahrwerkes mit dem lokalen Multiroboter-Koordinatensystem $\Sigma_{R,m}$ dargestellt. Die lokalen Robotermittelpunkte $\mathbf{p}_{0,1}$ und $\mathbf{p}_{0,2}$ des Master- und des Slaveroboterfahrwerkes wurden so definiert, dass sie schon im Multirobotermittelpunkt $\mathbf{p}_{0,m}$ liegen.

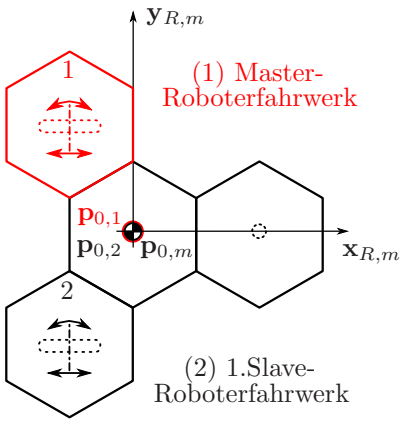


Abbildung 7.11: Multiroboterfahrwerk „Vollständig verteilter Fall“ - Testfahrt 3

In Abbildung 7.12 ist der abzufahrende Sollpfad für das Multiroboterfahrwerk dargestellt. Das rote Kreuz legt dabei den Startpunkt des Multiroboterfahrwerkes fest und der rote Pfeil zeigt die Richtung an, in der das Multiroboterfahrwerk den vorgegebenen Sollpfad abfährt. Dieser Pfad wird vom Multiroboterfahrwerk im Fahrmodus „Eyecatcher“ abgefahren. Dabei zeigt die $\mathrm{x}_{R,m}$-Achse des Multiroboterfahrwerkes während der Fahrt immer auf den raumfesten Punkt

$$\mathbf{p}_{eye} = \begin{bmatrix} 1 \\ -0.5 \end{bmatrix}, \tag{7.1}$$

der als blauer Stern in Abbildung 7.12 dargestellt ist. Der aufgrund der Odometriedaten

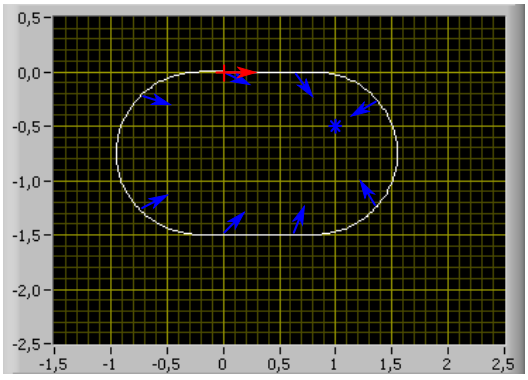


Abbildung 7.12: Vorgegebener Sollpfad - Testfahrt 3

(Abb. 7.14 und 7.15) errechnete abgefahrene Istpfad, wird in Abbildung 7.13 mit dem vorgegeben Sollpfad verglichen. Es ist zu erkennen, dass beim Multiroboterszenario „Vollständig verteilter Fall“ die Abweichungen zwischen Soll- und Istpfad deutlich größer sind, als bei den vorher durchgeführten Testfahrten 1 und 2. Der Grund liegt darin, dass beim „Vollständig verteilter Fall“ die Odometriedaten von zwei Fahrwerksrädern bzw. von zwei Roboterfahrwerken für die Berechnung der aktuellen Multiroboterposition benötigt werden. Daher muss für die Berechnung der aktuellen Muliroboterposition das Slaveroboterfahrwerk seine Odometriedaten über das WLAN-Netzwerk an den Masterroboter senden. Da es bei dieser Datenübertragung öfters zu Verzögerungen (siehe Abschnitt 5.2.4) kommt, empfängt das Masterroboterfahrwerk die aktuellen Odometriedaten des Salveroboterfahrwerkes nicht zeitgerecht. Dadurch kommt es zu einer fehlerhaften Berechnung der Multiroboterposition, die sich wieder negativ auf die Berechnung des nächsten Multiroboter-Fahrbefehls auswirkt. Daher ergibt sich während der Fahrt eine größere Abweichnung vom Sollpfad. In Abbildung 7.14 und 7.15 sind die

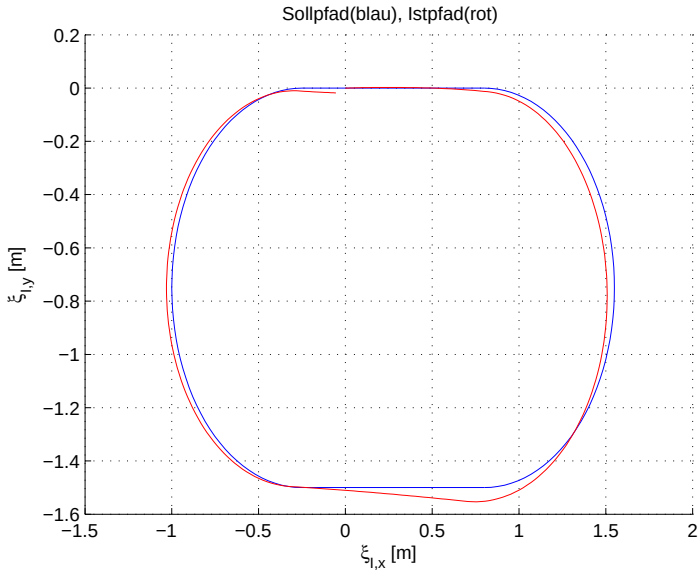


Abbildung 7.13: Vergleich von Soll- und Istpfad - Testfahrt 3

Radlenkwinkel- und Raddrehzahldaten der einzelnen Roboterfahrwerksräder grafisch dargestellt, die während der Fahrt des Multiroboterfahrwerkes aufgezeichnet wurden. Der Grund, warum die Raddrehzahl in Abbildung 7.15 in den ersten 0.8 Sekunden bei Null liegt ist, dass das Multiroboterfahrwerk solange mit der Abfahrt wartet, bis jede Radwabe ihre erste Radlenkwinkel-Sollvorgabe angefahren hat. Erst dann beginnt das

Multiroboterfahrwerk mit der Pfadabfahrt.

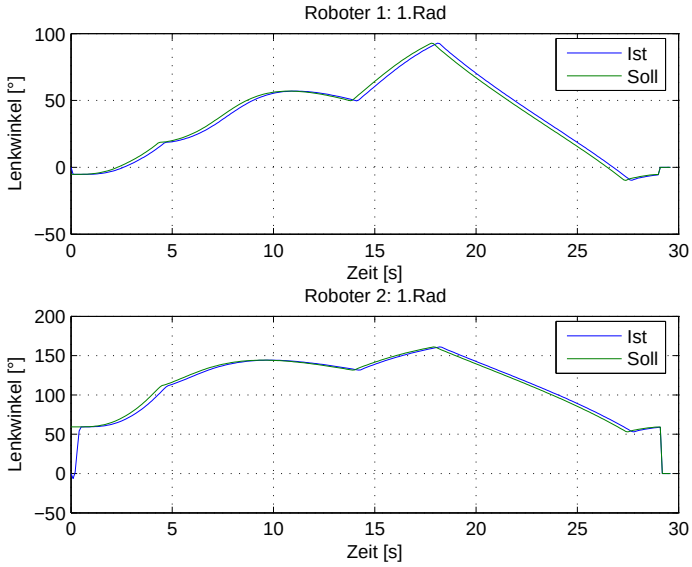


Abbildung 7.14: Radlenkwinkeldaten - Testfahrt 3

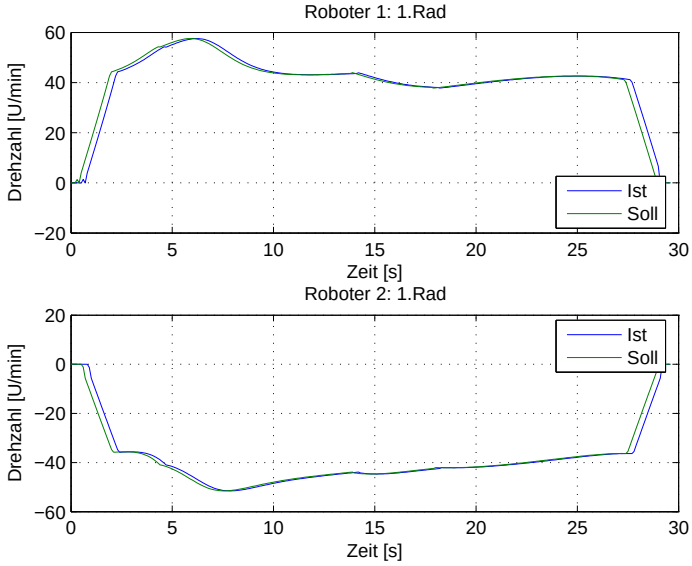


Abbildung 7.15: Raddrehzahldaten - Testfahrt 3

Fazit - Testfahrt 3:

Mit der Testfahrt 3 konnte gezeigt werden, dass die praktische Umsetzung der Multiroboterkonfiguration „Vollständig Verteilter Fall“ mit der entworfenen

Multirobotersoftware „MPRD V3.4“ prinzipiell funktioniert. Durch die fehlerhafte Positionsbestimmung des Multiroboter-Fahrwerkes, aufgrund von zu spät empfangenen Slave-Odometriedaten (siehe Abschnitt 5.2.4), ergibt sich leider eine größere Abweichung zum vorgegebnen Multiroboter-Sollpfad. Mit der Behebung des Übertragungsproblems sollten auch die großen Abweichungen vom vorgegebnen Sollpfad der Vergangenheit angehören.

Kapitel 8

Zusammenfassung und Ausblick

8.1 Zusammenfassung

Ziel dieser Diplomarbeit war es unter anderem, eine Interpolation zwischen den Radlenkwinkel-Sollvorgaben, die ca. alle 50 - 100 ms von der Recheneinheit (cRIO) an die Motorregelgeräte (ELMO-Whistles) erfolgen, zu erreichen. Die Interpolation zwischen den Radlenkwinkel-Sollwertvorgaben konnte über die Positon-Time-Tables (Punkt 2.3.2) der Motorregelgeräte „ELMO Whistles“ umgesetzt werden. Dadurch konnte das Rattern der Radlenkeinheiten während der Fahrt des Roboterfahrwerkes [14], aufgrund von sprunghaften Änderungen in den Radlenkwinkel-Sollwertvorgaben, vollständig eliminiert werden. Der Nachteil den die Interpolation (3. Ordnung) mit sich bringt ist, dass zwischen der Vorgabe und dem Anfahren eines Radlenkwinkel-Sollwertes drei Berechnungsperioden T_d vergehen. Durch diese Verzögerung ergibt sich bei der aktuellen Umrechnung (siehe Glg. 4.4 - Θ aus Odometriedaten) vom globalen $\dot{\boldsymbol{\xi}}_I$ in den lokalen Roboterfahrbefehl $\dot{\boldsymbol{\xi}}_R$ ein Fehler, der für die Abweichungen zwischen Soll- und Istpfad mitverantwortlich ist.

In Kapitel 3.2 wurde ein Algorithmus entworfen und vorgestellt, mit dem die Modelldefinition eines Roboterfahrwerkes und infolgedessen auch das kinematische Robotermodell automatisch bestimmt werden kann. Vorteil dieser autonomen Bestimmung der Modelldefinition ist, dass sich der Anwender nicht mehr um die Aktualisierung der Modelldefinitionsdaten kümmern muss, wenn die Konfiguration des Roboterfahrwerkes, durch hinzufügen oder entfernen von Radwaben, geändert wird. Der Nachteil der in Kapitel 3.2 vorgestellten Methode zur autonomen Bestimmung

der Modelldefinition, liegt im enormen Hardwareaufwand. Für die Funktion dieses Algorithmus muss jede Wabe des Roboterfahrwerkes mit einer Recheneinheit ausgestattet werden. Der Benutzer muss je nach Anwendungsfall entscheiden, ob der erhöhte Hardwareaufwand gerechtfertigt ist oder nicht.

Voraussetzung für die korrekte Pfadabfahrt eines Multiroboterfahrwerkes unter der entworfenen Multirobotersoftware „MPRD V3.4“ (Kap. 6) ist, dass vor Fahrtbeginn die Systemuhren der Motorregelgeräte „ELMO-Whistles“ an die Systemuhr der zugehörigen Recheneinheit „cRIO“ und die Systemuhren der Slave-cRIOs an die Systemuhr des Master-cRIOs synchronisiert worden sind. Die Synchronisation der Motorregelgeräte „ELMO-Whistles“ mit der Recheneinheit „cRIO“ konnte ohne Probleme über das SYNC-Timestamp-Verfahren (Punkt 5.1.1) durchgeführt werden. Bei der Synchronisation zwischen den Slave-cRIOs und dem Master-cRIO traten jedoch größere Schwierigkeiten auf. Da im Punkt 5.2 zwei von drei angeführten Synchronisationsmethoden nicht erfolgreich umgesetzt werden konnten, musste die Synchronisation der Recheneinheiten „cRIOs“ über den zeitgleichen „Boot Up“ der cRIO-Systeme durchgeführt werden. Es handelt sich dabei um eine Notlösung, bis eine der genannten Synchronisationsmethoden erfolgreich umgesetzt werden kann.

Die im Zuge dieser Diplomarbeit programmierte Multirobotersoftware „MPRD V3.4“ (Kap. 6) ist aus drei Hauptprogrammen aufgebaut. Das Programm „Host-Main.vi“ wird auf dem Host-PC ausgeführt, und dient als Anzeige- und Bedienterminal für die Multirobotersoftware bzw. das Multiroboterfahrwerk. Das Programm „Master-MainDriveComb.vi“ wird auf der Recheneinheit (cRIO) des Masterroboterfahrwerkes ausgeführt. Dieses Programm steuert das Masterroboterfahrwerk und sendet dabei die auf seine Ausführbarkeit hin überprüften Multiroboter-Fahrbefehle $\dot{\boldsymbol{\xi}}_m$ an die Slaveroboterfahrwerke. Das Programm „Slave-MainDriveComb.vi“ wird auf den Recheneinheiten der Slave-Roboterfahrwerke ausgeführt und steuert das jeweilige Slaveroboterfahrwerk.

Die in Kapitel 7 durchgeführten Testfahrten haben gezeigt, dass die in der Theorie ausgearbeiteten Multiroboter-Brechnungsvorschriften in der Multirobotersoftware „MPRD V3.4“ richtig umgesetzt werden konnten. Die Testfahrten haben aber auch die noch vorhandenen Schwachpunkte bei dieser Multiroboter-Softwareversion aufgezeigt, wie zum Bei-

spiel die Kommunikation zwischen den Roboterfahrwerken.

8.2 Ausblick

Damit ein vorgegebener Sollpfad von einem Multiroboterfahrwerk korrekt abgefahren werden kann, muss die Kommunikation zwischen dem Masterroboterfahrwerk und den Slaveroboterfahrwerken einwandfrei funktionieren. Die für die Netzwerkkommunikation zuständige STM-Library, die auf einem TCP/IP-Protokoll basiert, konnte diesem Anspruch leider nicht vollständig gerecht werden. Daher wird für die Zukunft empfohlen, dass bei zeitkritischen Daten, worunter auch die gesendeten Masterroboter-Fahrbefehle an die Slaveroboterfahrwerke fallen, ein UDP-Protokoll für den Datenaustausch verwendet werden sollte. Die Datenübertragung erfolgt beim UDP-Protokoll um einiges schneller als beim TCP/IP-Protokoll, jedoch auf Kosten der Übertragungssicherheit.

Mittlerweile wurde das auf der National Instruments Homepage zur Verfügung gestellte Synchronisationsprogramm „Software Synchronisation Beta Program“[1] durch ein neues Programm ersetzt. Der Name des Programms lautet „NI-TimeSync 1.0.1“[2] und kann ebenfalls über die Homepage von National Instruments bezogen werden. Da es sich dabei um keine Beta-Version mehr handelt, sollte eine Synchronisation der cRIO-Systemuhren mit diesem Programm erfolgreicher sein. Damit könnte die aktuell verwendete Synchronisations-Notlösung, durch den zeitgleichen „Boot Up“ der cRIO-Systeme, ersetzt werden.

Weiterführende Arbeiten könnten sich noch mit der Entwicklung eines Multi-Pfadplaners, für das Multiroboterszenario beschäftigen. Durch die Integration eines Multi-Pfadplaners in das Programm „Host-Main.vi“, könnte im Multiroboterfall für alle Roboterfahrwerke ein individueller Sollpfad vorgegeben werden. Dadurch wären die Roboterfahrwerke in der Lage, sich variabel bzw. unabhängig voneinander zu bewegen.

Wenn in der Multirobotersoftware keine Pfadregelung implementiert ist, dann wäre es von Vorteil, wenn der Winkel Θ, der für die Umrechnung (siehe Glg. 4.4) des globalen $\dot{\boldsymbol{\xi}}_I$ in den lokalen Roboterfahrbefehl $\dot{\boldsymbol{\xi}}_R$ benötigt wird, nicht aus den Odometriedaten gewonnen werden würde, sondern einfach aus der Sollpfadvorgabe abgeleitet werden würde.

[1] http://zone.ni.com/devzone/cda/tut/p/id/8278
[2] http://joule.ni.com/nidu/cds/view/p/id/2083/lang/en

Dadurch könnten auch bei integrierter Interpolation zwischen den Sollwertvorgaben, die Abweichungen zwischen Soll- und Istpfad noch verringert werden.

Anhang A

Anhang

A.1 Parametrierung - Motorregelgeräte ELMO Whistles

Die Motorregelgeräte „ELMO Whistles“ sind über CAN-Bus mit der Recheneinheit (cRIO) des Roboterfahrwerkes verbunden. Über diese CAN-Bus-Verbindung erfolgt der gesamte Datenaustausch zwischen den beiden Geräten sowie auch die Parametrierung der Motorregelgeräte „ELMO Whistles“. Da die Motorregelgeräte „ELMO Whistles“ auf dem CANopen-Standard basieren, müsste dieser hier im Detail erklärt werden. Da das aber den Rahmen dieser Diplomarbeit sprengen würde, wird für eine ausführliche Erklärung zu diesem Thema auf [8] verwiesen. Im Punkt A.1.1 werden ein paar Fachbegriffe zum Thema „CANopen“ kurz erläutert, da diese in den weiterführenden Punkten öfters vorkommen.

A.1.1 Begriffsklärung - CANopen

Die CANopen Anwendungsschicht legt verschiedene Kommunikationsobjekte fest, welche auf CAN-Telegramme abgebildet werden. CANopen entspricht dem definierten OSI-Referenzmodell (Open System Interconnection) und basiert auf dem seriellen Bussystem CAN (Controller Area Network), welches im Wesentlichen die Funktionalität der physikalischen Schicht und der Verbindungschicht abdeckt. Alle CANopen Funktionen werden auf mehreren CAN-Objekten (CBOs) abgebildet. Die CANopen-Spezifikationen[1] und CANopen-Empfehlungen enthalten weitergehende Definitionen, teilweise sind diese auch anwendungsspezifisch.

[1] www.can-cia.org

A.1.1.1 Prozessdatenobjekte (PDO)

Bei CANopen werden die Prozessdaten in Segmente zu maximal 8 Byte aufgeteilt. Diese Segmente heißen Prozessdatenobjekte(PDOs). Ein PDO-CAN-Telegramm besteht aus dem 11 Bit Identifier und bis zu 8 Datenbytes. Durch die Wertigkeit des Identifiers ist auch die Priorität der CAN Nachricht festgelegt. Je niederwertiger der Identifier desto höher ist die Priorität der Nachricht. Es wird zwischen Empfangs-PDOs (Receive-PDOs, RxPDOs) und Sende-PDOs (Transmit-PDOs, TxPDOs) unterschieden, wobei die Bezeichnung jeweils aus Gerätesicht erfolgt.

A.1.1.2 Servicedatenobjekte (SDO)

Für den Zugriff auf Einträge im Objektverzeichnis werden Servicedatenobjekte benutzt, dabei hat jedes CANopen Gerät mindestens ein SDO. Ein SDO errichtet eine peer-to-peer Kommunikationsverbindung zwischen zwei CANopen Geräten. Mit dem SDO-Protokoll können, im Vergleich zum PDO-Protokoll, mehr als 8 Byte Daten übertragen werden → „SDO Segment Protokoll“. Das SDO-Protokoll wird hauptsächlich verwendet um Konfigurationsdaten zwischen zwei CANopen Geräten auszutauschen, neue Firmware auf ein Gerät zu laden oder das PDO-Mapping zu verändern. Servicedatenobjekte haben eine geringere Priorität als PDOs.

A.1.2 „Operational-State“ - Motorregelgeräte ELMO Whistles

Wird auf die Motorregelgeräte „ELMO-Whistles“ die Versorgungsspannung aufgeschalten, läuft als erstes die „Boot-Up“ Phase, bei der die systemeigene Initialisierung der Whistle-Hardware vorgenommen wird. Nach dem Boot-Up befindet sich das Whistle im so genannten „Pre-operational-State“ bei dem das Whistle nur auf Service-Daten-Objekte (SDOs) und auf Network Management Messages (NMT) antwortet. Dieser Programm-Modus kann zum Beispiel verwendet werden, um neue Software auf das Gerät aufzuspielen. Da aber die Kommunikation mit dem Whistle fast ausschließlich über „Simple IQ Kommandos“ [5] erfolgt, abgesehen vom CAN-Mapping (Punkt A.1.4), muss das Whistle auch auf Prozess-Daten-Objekte (PDOs) reagieren. Damit das der Fall ist, muss vom „Pre-operational-State“ in den „Operational-State“ gewechselt werden. Hierfür wird nach dem Boot-Up des Whistles das NTM-Kommando „Start remote node“ gesendet. Ein NMT-Kommando (siehe [4] Kapitel 7-1) ist immer 2-Byte lang und besteht aus dem Identfier, Command Specifier und der Node-ID. In Tabelle A.1 ist das NMT-Kommando „Start remote node“ in hexadezimaler Darstellung angegeben. Node ID = 0 bedeutet, dass sich alle

Whistles (Nodes) die mit dem CAN-Bus verbunden sind, angesprochen fühlen.

Identifier	Command Specifier	Node-ID
0x000	0x01	0x00

Tabelle A.1: NMT-Kommando: „Start remote Nodes"

Nachdem sich alle Motorregelgeräte im „Operational-State" befindet, können alle notwendigen Grundeinstellungen vorgenommen werden.

A.1.3 Grundeinstellungen - Motorregelgeräte ELMO Whistles

Die Grundeinstellung der ELMO-Whistles dient dazu, um die positive Drehrichtung der angeschlossenen BLDC-Motoren (siehe 2.3.1), die Maximalwerte der Wicklungsströme, die Auflösung des an die Motoren angeflanschten optischen Inkrementaldrehgebers, die maximale Beschleunigung beim Anfahren einer neuen Sollposition und die maximale Verzögerung beim Abbremsen festzulegen. Zusätzlich müssen noch allgemeine Parameter wie Abtastzeit des Stromregelkreises und Kommunikationsparameter eingestellt werden. Damit die Einstellungen nicht jedes Mal händisch eingegeben werden müssen, wurden die festgelegten Einstellungen in einer Datei (Application-File) gespeichert, die auf die ELMO-Whistles geladen werden kann. Es stehen zwei Dateien zur Auswahl, wobei eine für den Radlenkmotor(app-steer-ts-100μs.dat) und die andere für den Motor des Radantriebes (app-drive-ts-100μs.dat) gedacht ist. Die Application-Files können über die serielle Schnittstelle RS232 oder über den CAN-Bus der ELMO-Whistles geladen werden. Kommunikationsparameter wie die Node-ID müssen für jedes ELMO-Whistle separat eingestellt werden. Diese Einstellung kann am einfachsten mit dem Programm „ELMO-Composer" vorgenommen werden. Eine genaue Beschreibung ist der Projektarbeit [8] zu entnehmen.

A.1.4 CAN mapping

Für die Bestimmung des aktuellen Momentanpoles eines Roboterfahrwerkes sowie für die Visualisierung der Radantriebsdaten werden von den Recheneinheiten (cRIOs) zu jedem neuen Berechnungszyklus T_d die aktuellen Raddrehzahl- und Radlenkwinkelmesswerte der Roboterfahrwerksräder benötigt. Diese Daten können die Recheneinheiten (cRIOs) via CAN-Bus von den Motorregelgeräten „ELMO-Whistles" abrufen. Nebenbei müssen die Recheneinheiten (cRIOs) für die Synchronität zwischen ihnen und den ELMO-Whistles sorgen. Hierfür muss zu bestimmten Zeitpunkten ein CAN-SYNC-Kommando mit dem

darauf folgenden Timestamp gesendet werden. Ohne die Verwendung von CAN-Mapping muss ein CAN-Frame gesendet werden, der in Abbildung A.1 dargestellt ist.

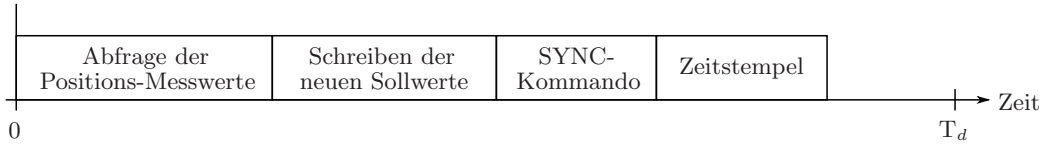


Abbildung A.1: CAN-Frame ohne „Mapping“

Bei Verwendung des CAN-Mappings kann das Verhalten der ELMO-Whistles so eingestellt werden, dass sie bei Erhalt eines CAN-SYNC-Kommandos mit den aktuellen Positionswerten der Inkrementaldrehgeber der Radlenk- und Radantriebsmotoren antworten. Dadurch erspart man sich die zusätzliche Abfrage der aktuellen Positionswerte, wodurch der zu sendende CAN-Frame kürzer wird. Damit die Motorregelgeräte „ELMO-Whistles“ die vorhin genannte Funktionalität erreichen, müssen dazu folgende CANopen-Objekte gemapped werden (siehe Tabelle A.2).

Identifier	**Databyte 1 - 8 (hex)**								**Description**
0x600	23	00	1A	00	00	00	00	00	Stop all emissions of TPDO1
0x600	23	00	1A	01	20	00	64	60	Objekt 0x6064 first 4 bytes (PDO)
0x600	23	00	1A	02	10	00	11	2F	Object 0x2f11 next 2 bytes (PDO)
0x600	23	00	1A	03	10	00	12	2F	Object 0x2f12 next 2 bytes (PDO)
0x600	23	00	18	02	01	00	00	00	„Transmit every SYNC“
0x600	23	00	1A	00	03	00	00	00	Activate the 3 mapped objects

Tabelle A.2: CAN-Mapping

Das Objekt „Transmit PDO mapping“ (Index 0x1A00, Subindex 0) definiert, wie viele anderen Objekte (z.B.: Positionsdaten, Zeigerstände, usw.) an ein einzelnes gesendetes PDO „angehängt“ werden können. Maximal können bis zu 8 Objekte „angehängt“ werden. Es muss jedoch darauf geachten werden, dass mit einem einzeln gesendeten PDO maximal 8 Datenbytes (64 Bit) übertragen werden können. In Tabelle A.2 wird in der ersten Zeile das Objekt „Transmit PDO mapping“ Subindex 0 auf 0 gesetzt. Das bedeutet, dass kein Objekt mehr an das PDO1 angehängt wird. Dadurch werden für die weiteren Einstellungen alle PDO1 Sendevorgänge des Motorregelgerätes „ELMO-Whistle“ gestoppt. In der zweiten Zeile wird das Objekt „Position actual value“ (Integer-32-Bit-Wert) mit dem Index 0x6064 an die ersten 4 Datenbytes des PDO1 gehängt. In der dritten und vierten Zeile werden noch zusätzlich die Objekte „PVT head pointer“ (Index 0x2f11) und „PVT tail pointer“ (Index 0x2f12) an das PDO1 gehängt. Diese letzten beiden Objekte dienen nur

zu Kontrolle, wo sich gerade der Schreib- und Lesezeiger der „Position Time Table (PTT)“ befindet. Mit diesen drei, an das PDO1 angehängte Objekt, sind nun alle 8 Datenbytes belegt. Das Objekt „Transmit PDO communication parameter“ (Index 0x1800, Subindex 2) legt den Übertragungs-Typ des PDO1 fest. Wird er auf den Wert 1 gesetzt, so sendet das ELMO-Whistle bei jedem erhaltenen SYNC-Kommando, das vorhin definierte PDO1. In Zeile fünf ist das zu sendende SDO angegeben. Um nun die gemappedten Objekte zu aktivieren, wird das Objekt „Transmit PDO mapping“ Subindex 0 auf 3 gesetzt. Somit besitzt das ELMO-Whistle das gewünschte Verhalten. Zu jedem empfangenen SYNC-Kommando antwortet es mit einem PDO, das die aktuellen Positionsdaten, den „PVT head pointer“ und den „PVT tail pointer“ beinhaltet. Bei der Verwendung von CAN-Mapping ist der gesendete CAN-Frame in Abbildung A.2 dargestellt.

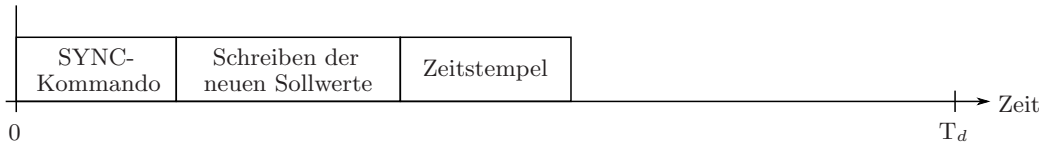


Abbildung A.2: CAN-Frame mit „Mapping“

A.1.5 Grundeinstellungen - Position Time Table (PTT)

Bevor die Position-Time-Table (Punkt 2.3.2) gestartet werden kann, muss sie parametriert werden und mit mindestens drei absoluten Sollpositionwerten beschrieben sein, da diese für die Berechnung des Interpolationspolynomes 3.Ordnung benötigt werden. In der Tabelle A.3 sind die für die Parametrierung nötigen „Simple IQ“-Befehle [5] mit den dazugehörigen Werten angegeben.

„Simple IQ“-Befehl	**Beschreibung**
MO = 0	Motor abschalten
PX = 0	Position Counter auf Null setzten
MP[1] = 1	Erste gültige Zeile in der PTT
MP[2] = 10	Letzter gültige Zeile in der PTT
MP[3] = 1	PTT wird zyklisch abgearbeitet, d.h 10→1
MP[4] = 250	$T_d/T_{d,Positions-RK}$ = 100ms/0,4ms = 250
QP[1..3] = 0	Ersten 3.Werte der PTT auf Null setzten
PT = 1	Startpunkt in der PTT
MO = 1	Motor wieder freigeben

Tabelle A.3: Motion-Parameter-PTT

Um alle notwendigen Parameter für die PTT zu setzten, muss als erstes der Motor abgeschaltet werden. Als nächstes wird der Zähler des Drehinkrementalgebers auf Null gesetzt, damit es zu keinem Positions-Offset kommt. Über den Parameter „MP[2]“ wird die Größe der PTT festgelegt, die wahlweise mit 10 Einträgen festgelegt wurde. Über den Parameter MP[3] wird festgelegt, ob die PTT zyklisch abgearbeitet werden soll oder ob sie beim letzten Eintrag beendet werden soll. Damit das Roboterfahrwerk auch längere Pfade abfahren kann, muss die PTT aufgrund ihrer Größe zyklisch abgearbeitet werden. Das heißt, wenn der 10. Positionswert gelesen wurde, dann springt der Lesezeiger wieder auf den 1. Tabellenplatz. Dabei muss sichergestellt werden, dass keine alten Sollpositionswerte eingelesen werden, indem der 11. Sollpositionswert zeitgerecht wieder auf den 1. Tabellenplatz geschrieben wird. Der Parameter MP[4] legt das Verhältnis zwischen der Abtastperiode der PTT und der Abtastperiode des Positionsregelkreises fest. Da die Abtastperiode des Stromregelkreises auf 100 μs festgelegt wurde, ergibt sich für den Positionsregelkreis eine Abtastperiode von 400 μs (4-fache Abtastperiode des Stromregelkreises). Soll zum Beispiel die PTT mit 100 ms abgearbeitet werden, dann erhält man für den Parameter MP[4] einen Wert von 250 (max. 256! siehe [5]). Anschließend werden noch die drei ersten Tabelleneinträge der PTT auf Null gesetzt, da die Sollvorgaben erst zu Laufzeit berechnet werden. Der Parameter „PT“ definiert bei dem nächst erhaltenen „Begin Motion“ (BG) oder „Begin Motion at Defined Time“ (BT=Value), den Startpunkt in der PTT. Da die PTT mit dem ersten Eintrag starten soll, wird der Parameter „PT“ auf Eins gesetzt. Nun wurden alle notwendigen Einstellungen für die PTT getroffen und somit kann der Motor wieder freigegeben werden. Bei dem nächst erhaltenen BG- oder BT-Kommando [5] wird die PTT gestartet und die eingetragenen absoluten Positionswerte angefahren.

A.2 Qualitative Roll- und Gleitbedingungen eines Standardrades

Über die Tabelle A.4 können die qualitativen Roll- und Gleitbedingungen eines Standardrades, aufgrund der polaren Abstandskoordinaten l_i und α_i zum Robotermittelpunkt $\mathbf{p}_{0,i}$ hin, bestimmt werden.

Betriebsmodus	$\mathbf{j_q}$	$\mathbf{c_q}$
OK: Antrieb und Lenkung funktionieren	$\begin{bmatrix} \sin(\alpha) & \sin(\alpha+\frac{\pi}{2}) \\ -\cos(\alpha) & -\cos(\alpha+\frac{\pi}{2}) \\ -l\cdot\cos 0 & -l\cdot\cos\frac{\pi}{2} \end{bmatrix}^T$	$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}^T$
Fehler 1: Antrieb funktioniert, aber blockierte Lenkung	$\begin{bmatrix} \sin(\alpha+\beta) \\ -\cos(\alpha+\beta) \\ -l\cdot\cos\beta \end{bmatrix}^T$	$\begin{bmatrix} \cos(\alpha+\beta) \\ \sin(\alpha+\beta) \\ l\cdot\sin\beta \end{bmatrix}^T$
Fehler 2: Frei drehendes Rad und blockierte Lenkung	$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}^T$	$\begin{bmatrix} \cos(\alpha+\beta) \\ \sin(\alpha+\beta) \\ l\cdot\sin\beta \end{bmatrix}^T$
Fehler 3: Blockiertes Rad und funktionierende Lenkung	$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}^T$	$\begin{bmatrix} \cos(\alpha) & \cos(\alpha+\frac{\pi}{2}) \\ \sin(\alpha) & \sin(\alpha+\frac{\pi}{2}) \\ l\cdot\sin 0 & l\cdot\sin\frac{\pi}{2} \end{bmatrix}^T$
Fehler 4: Frei drehendes Rad und funktionierende Lenkung	$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}^T$	$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}^T$

Tabelle A.4: Qualitative Roll- und Gleitbedingungen eines Standardrades mit verschiedenen Fehlerfällen

Literaturverzeichnis

[1] Bishop, Robert H.: *Labview 8 Student Edition.* Pearson, 2007

[2] Bunse, Wolfgang ; Bunse-Gerstner, Angelika: *Numerische Lineare Algebra.* Stuttgart. Teubner, 1985

[3] Elmo Motion Control Ltd.: *Simpl IQ Software Manual.* Ver. 1.1, September 2004

[4] Elmo Motion Control Ltd.: *CANopen DS 301 Implementation Guide.* Ver. 2.1, August 2008

[5] Elmo Motion Control Ltd.: *Simpl IQ - Command Reference Manual.* V 4.2, August 2008

[6] Geunho Lee, Nak Young C.: Decentralized formation control for small-scale robot teams with anonymity. In: *Mechatronics* (2008)

[7] Jantscher, Simon: *Modulares, rekonfigurierbares Roboterfahrwerk*, Technische Universität Graz, Diplomarbeit, 2009

[8] Kalss, Alexander: *Erstellung eines Labview VIs zur Kommunikation zwischen NI-cRIO und ELMO - DUO Motion Control Units mittels CAN-BUS*, Technische Universität Graz, Projektarbeit, 2009

[9] Köb, Johannes: *Modellbasierte Regelung eines Roboterfahrwerkes*, Technische Universität Graz, Diplomarbeit, 2005

[10] National Instruments: *LabVIEW Basics 2: Entwicklung, Kurshandbuch*, September 2007

[11] National Instruments: *LabVIEW Basics I: Einführung, Kurshandbuch*, November 2007

[12] National Instruments: *LabVIEW Real-Time Application Development Exercises*, February 2009

[13] Schörghuber, Christoph: *Dockingmechanismus für einen modularen mobilen Roboter*, Technische Universität Graz, Projektarbeit, 2008

[14] Schörghuber, Christoph: *Automatisierung von rekonfigurierbaren Multi-Roboter-Systemen*, Technische Universität Graz, Diplomarbeit, 2009

[15] Siegwart, R. ; Nourbakhsh, I.R.: *Introduction to Autonomous Mobile Robots*. MIT Press, 2004

[16] Zassenhaus, Hans: *Ueber einen Algorithmus zur Bestimmung der Raumgruppen*. 1948

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am ……. ……..
(Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

……. ……..
date (signature)