Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Diplomarbeit

Bang Giang Nguyen

Entwicklung von Bibliotheksfunktionen und Steuerprogrammen für einen Roboter

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Bang Giang Nguyen

Entwicklung von Bibliotheksfunktionen und Steuerprogrammen für einen Roboter

Diplomarbeit eingereicht im Rahmen der Diplomprüfung
im Studiengang Informations- und Elektrotechnik
Studienrichtung Automatisierungstechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Ing. Andreas Suhl
Zweitgutachter : Prof. Dr. Ing. Wilfried Wöhlke

Abgegeben am 14. Oktober 2010

Bang Giang Nguyen

Thema der Diplomarbeit

Entwicklung von Bibliotheksfunktionen und Steuerprogrammen für einen Roboter

Stichworte

Steuerungstechnik, Robotik, Sensor, Aktor, Ereignisdiskretes System, Modellierung, Petrinetz, Kanonische Beschreibungsform und Grafcet.

Kurzzusammenfassung

Im Rahmen dieser Arbeit sollen intelligente Sensoren/Aktoren für einen Roboter entwickelt werden, damit er über neue Funktionen verfügt um komplexe Aufgaben auszuführen. Zum Prüfen neuer Funktionen ist ein Steuerprogramm als Testaufgabe zu erstellen, durch dieses alle neue Funktionen und die Zusammenarbeit zwischen dem Roboter und einer Steuerung getestet werden sollen. Wichtig ist, dass der Roboter sowohl die Aufgabe erledigt, als auch Fehler und Ausnahmen erkennen und darauf reagieren kann.

Ende des Textes

Bang Giang Nguyen

Title of the paper

Development of library functions and control programs for a robot

Keywords

Control technology, Robotics, Sensor, Actuator, Discrete event system, Modelling, Petri net, Canonical form of description and Grafcet

Abstract

As part of this work is the designing of intelligent sensors/actuators for a robot, so he has new functions to perform complex tasks. For testing new features a control program is created as test task, through it all the new features and cooperation between the robot and a controller should be tested. Important is that the robot completes the task, as well as recognizes and responds to errors and exceptions.

End of text

Inhaltsverzeichnis

Verzeichnis der Abbildungen

Verzeichnis der Tabellen

Verzeichnis der Codeauszüge

1 Einleitung

Im 20. Jahrhundert entwickelte sich der Bereich Robotik weltweit sehr schnell. Robotersteuerung wird in vielen Feldern angewendet. In einigen Bereichen sind Roboter heutzutage sogar nicht mehr wegzudenken, zum Beispiel in der Fertigung von Mikrocontrollern oder der Automobilindustrie. Derzeit ist der größte Einsatzbereich für Roboter die Fertigungsindustrie. Nach Informationen der IFR (International Federation of Robotics) beträgt die Anzahl der Industrieroboter, die im Jahr 2009 weltweit in Betrieb waren, ca. 1 Million.

Zusammen mit den hardwaretechnischen Einrichtungen spielen die softwaretechnischen Komponenten im Industrieroboter eine wichtige Rolle. Folgende Elemente sind nicht aus dem Robotersystem zu werfen:

- Steuerung
- Intelligente Sensoren/Aktoren
- Steuerprogramm (Automatenmodell)

Im Rahmen dieser Diplomarbeit sollen neue Bibliotheksfunktionen (intelligente Sensoren/Aktoren) und Steuerprogramme für einen Roboter realisiert werden. Danach sind die neuen softwaretechnischen Komponenten zusammen mit dem Roboter im Betrieb zu untersuchen. Die Untersuchungen sollen auf der Grundlage ereignisdiskreter Modelle erfolgen.

In dieser Arbeit werden zuerst die wichtigen Begriffe und Grundlagen der Steuerungstechnik und Robotik vorgestellt und beschrieben. Anschließend wird die Problem- und Aufgabenstellung der Arbeit dargestellt. Danach werden die speziellen Anwendungen, die für dieses Untersuchungssystem verwendet werden, vorgestellt. Darauf folgt die Beschreibung der Hardware- und Software-Komponenten des experimentellen Roboters. Danach werden die neuen intelligenten Sensoren/Aktoren und ihre Funktionalität dargestellt. Die Roboterdemonstration wird danach in einer experimentellen Untersuchung durchgeführt. Am Ende werden die Ergebnisse der ganzen Arbeit zusammengefasst.

2 Grundlagen der Steuerungstechnik und Robotik

2.1 Steuerungstechnik

Um den Begriff Steuerungstechnik besser zu verstehen, muss der Oberbegriff Automatisierungstechnik zuerst begriffen werden. Wie Lothar Litz definiert hat:

„Durch Automatisierung werden dynamische Prozesse in ihrem Verlauf erfasst und derart gezielt beeinflusst, dass sie vorgegebene Aufgaben und Funktionen selbsttätig erfüllen.“ [4]

Die Automatisierungstechnik beschäftigt sich mit der Überwachung und Steuerung technischer Systeme. Durch Automatisierungseinrichtungen sollen Geräte und Anlagen so überwacht und gesteuert werden, dass sie ihre Funktionen weitgehend selbsttätig erfüllen und Sicherheitsanforderungen genügen. In Erweiterung dessen will man zukünftig sogar erreichen, dass automatisierte Systeme die aktuell zu erfüllenden Aufgaben selbstständig erkennen und daraus die auszuführenden Lösungsschritte ableiten, d.h. autonom (ohne den Eingriff eines Menschen) arbeiten. Es ist das Ziel der Automatisierungstechnik, Systeme so zu steuern, dass sie selbständig arbeiten [5].

Automatisierungstechnik umfasst sowohl Regelungstechnik als auch Steuerungstechnik. Bei der Regelungstechnik handelt es sich um die gezielte Beeinflussung dynamischer Prozesse. Bei diesen Prozessen sind die gemessenen bzw. verwendeten Signale kontinuierlich oder quasikontinuierlich. Bei der Regelung befindet sich das betrachtete System in einem geschlossenen Wirkungskreis. Abbildung 2.1 stellt die Struktur eines Regelkreises dar.

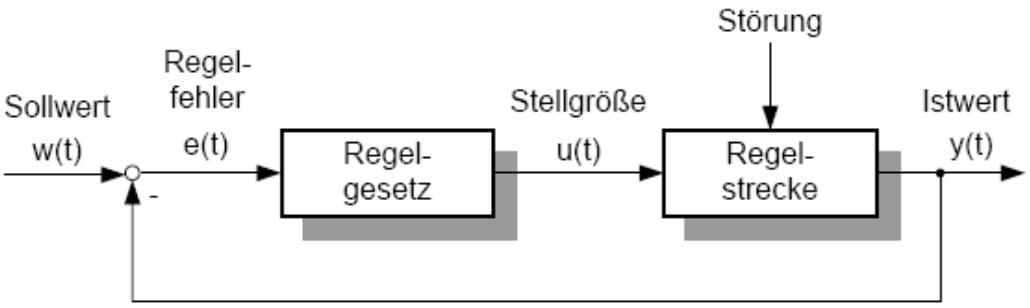


Abbildung 2.1 Strukturbild des Regelkreises [8]

Im Unterschied zur Regelung beeinflusst eine Steuerung das entsprechende System in einer offenen Wirkungskette. Gemäß DIN 19266 wird Steuerung wie folgt definiert:

„Das Steuern, die Steuerung, ist ein Vorgang in einem System, bei dem eine oder mehrere Größen als Eingangsgrößen andere Größen als Ausgangsgrößen aufgrund der dem System eigentümlichen Gesetzmäßigkeiten beeinflussen. Kennzeichen für das Steuern ist der offene Wirkungsweg oder ein geschlossener Wirkungsweg, bei dem die durch die Eingangsgrößen beeinflussten Ausgangsgrößen nicht fortlaufend und nicht wieder über dieselben Eingangsgrößen auf sich selbst wirken. Anmerkung: Die Benennung Steuerung wird vielfach nicht nur für den Vorgang des Steuerns, sondern auch für die Gesamtanlage verwendet, in der die Steuerung stattfindet."

Laut der Definition oben gibt es zwei wichtige Unterschiede zu der der Regelung. Zum einen wird bei einer Steuerung generell vom Mehrgrößenfall ausgegangen. Zum anderen existiert keine fortlaufende Beeinflussung der zu steuernden Größe über die Eingangsgrößen auf sich selbst [2].

Im Gegensatz zur Regelung arbeitet man beim Entwurf einer Steuerung immer noch häufig ohne ein mathematisches Modell der Strecke. Dafür muss erheblich mehr Aufwand in die Formulierung der Anforderungen und die daraus abgeleitete Modellierung der Steuerung selbst investiert werden. Das Modell der Steuerung erlaubt neben der Simulation umfangreiche Analysen, die beim – leider oft anzutreffenden – direkten Umsetzen der Anforderungen in eine Programmiersprache nicht möglich sind. Abbildung 2.2 zeigt die Struktur des Steuerkreises.

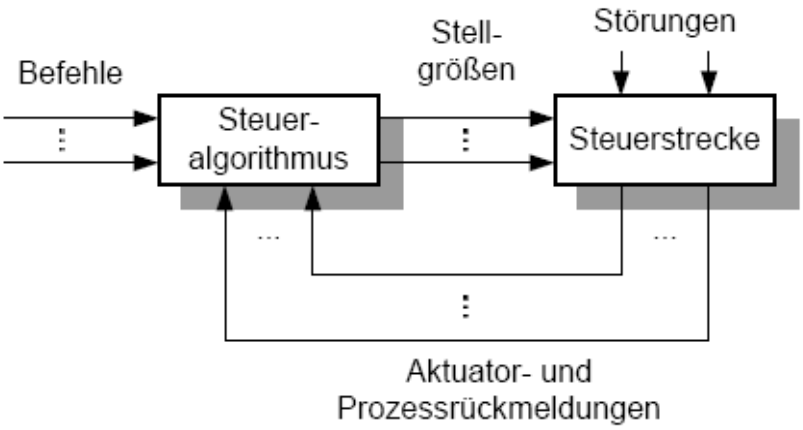


Abbildung 2.2 Strukturbild des Steuerkreises [8]

Bei Steuerungen unterscheidet man grundsätzlich zwischen Verknüpfungs- und Ablaufsteuerungen, die in den folgenden Abschnitten beschrieben werden.

2.1.1 Verknüpfungssteuerungen

Nach DIN 19226-5 wird Verknüpfungssteuerung wie folgt definiert: „Eine Verknüpfungssteuerung ordnet den Zuständen der Eingangssignale durch Boolesche Verknüpfungen definierte Zustände der Ausgangssignale zu. Auch Steuerungen mit Verknüpfungsgliedern und einzelnen Speicher- und Zeitfunktionen ohne zwangsläufig schrittweise Ablauf werden ebenso benannt."

Eine Verknüpfungssteuerung reagiert demgemäß bei einer bestimmten Belegung ihrer Eingangssignale mit einer definierten Belegung der Ausgangssignale [2]. Da mit einer solchen Zuordnung nur sehr einfache Probleme zu lösen sind, werden Ergänzungen um Zeit- und Speicherglieder zugelassen. Außerdem müssen Verknüpfungen in Echtzeit ausgewertet und ausgeführt werden.

2.1.2 Ablaufsteuerungen

Nach DIN 19226-5 lautet die Definition der Ablaufsteuerung: „Eine Ablaufsteuerung ist eine Steuerung mit zwangsläufig schrittweise Ablauf, bei der der Übergang von einem Schritt auf den oder die programmgemäß folgenden abhängig von Übergangsbedingungen erfolgt. Die Schrittfolge kann in besonderer Weise programmiert sein, z.B. mit

Sprüngen, Schleifen, Verzweigungen. Die Schritte der Steuerung entsprechen meist den prozessbedingt aufeinander folgenden Zuständen der zu steuernden Anlage."

Viele Probleme der Fertigungs- und Verfahrenstechnik stehen mit zeitlichen Abläufen in Verbindung und lassen sich als Folge von Schritten beschreiben. Dafür ist die Ablaufsteuerung geeignet. Sie ist gut für zeitliche, parallele oder zu synchronisierende Abläufe.

2.2 Robotertechnik

2.2.1 Einführung in Industrieroboter

Unter dem Begriff „Industrieroboter" werden nach heutigem Sprachgebrauch Handhabungsgeräte verstanden, die in mehreren Bewegungsachsen frei programmierbar mit Greifer oder Werkzeugen ausgerüstet sind und nach einem Programm ggf. unter Sensor-Kontrolle Handhabungs-, Prüf- oder Fertigungsoperationen durchführen können [9]. Die VDI-Richtlinie 2960 grenzt gegenüber Manipulatoren oder ähnlichen Betriebsmitteln die Industrieroboter weitgehend ab mit der Definition:

„Industrieroboter sind universell einsetzbare Automaten mit mehreren Achsen, deren Bewegungsmöglichkeiten i.a. durch einen oder mehreren Arme realisiert werden und die am Ende mit weiteren Gelenken (Nebenachsen) ausgerüstet sein können. Ihre Bewegungen müssen hinsichtlich Bewegungsfolge und Wegen bzw. Winkeln ohne mechanischen Eingriff in die Steuerung programmierbar sein. Industrieroboter sind mit Greifer, Werkzeugen, Messmittel oder Fertigungsmitteln ausrüstbar und können Handhabungs- und Fertigungsverfahren ausführen."

Bei einem Industrieroboter wirken unterschiedliche Komponenten aus den Bereichen des Maschinenbaus, der Elektrotechnik und der Elektronik zusammen. Im Verbund in einem flexiblen Fertigungssystem (Abbildung 2.3), bestehend aus mehreren meist numerisch gesteuerten, miteinander lose verketteten Einzelmaschinen, ist der Industrieroboter imstande, aufgrund der materialfluss- und steuerungstechnischen Verknüpfung automatisch Werkstücke in mittleren und kleinen Losgrößen bis zur Losgröße 1 zu bearbeiten [11].

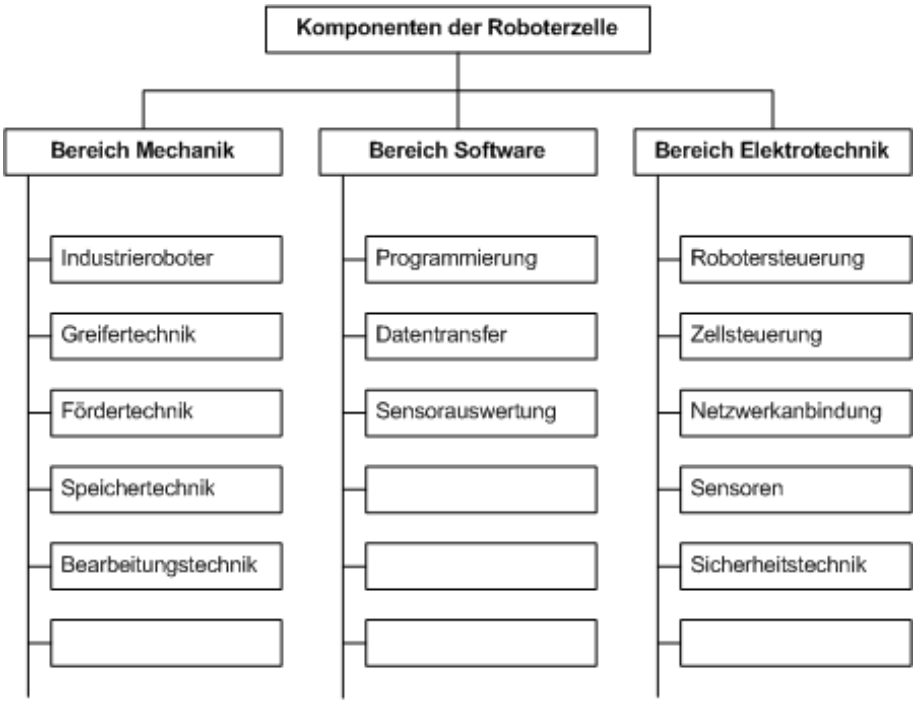


Abbildung 2.3 Komponenten der Roboterzelle [11]

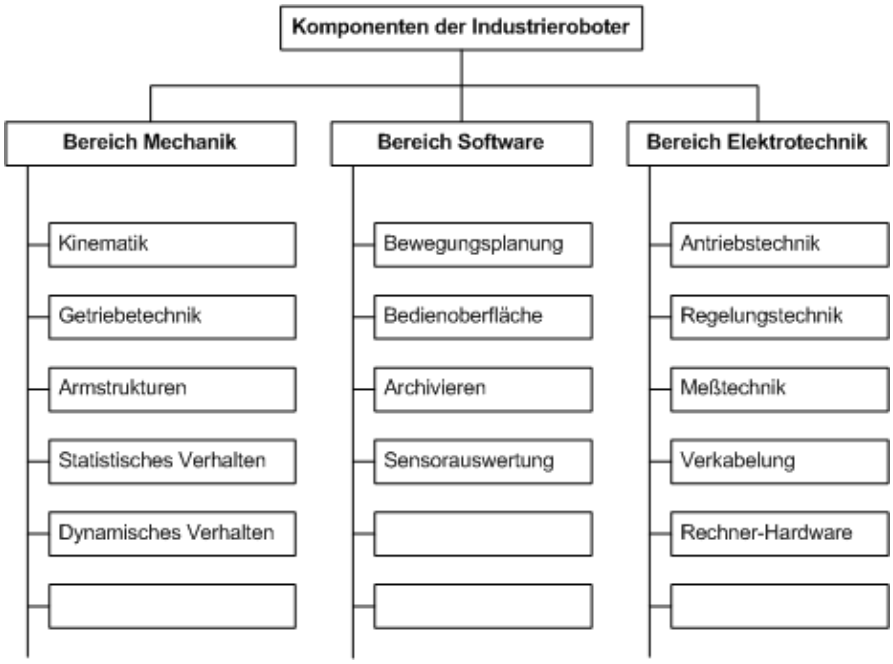


Abbildung 2.4 Komponenten der Industrieroboter [11]

Die Hauptanwendungsgebiete liegen zurzeit beim Schweißen, Lackieren, Beschichten und in der Montage. Zunehmend finden Industrieroboter jedoch auch beim Prüfen und bei Bearbeitungsverfahren Anwendung, wobei der Weiterentwicklung der Programmiertechnik und Sensorik Schrittmacherfunktion zukommt.

2.2.2 Roboterkinematiken

Die Hauptaufgabe des Teilsystems Kinematik ist das Herstellen der räumlichen Zuordnung zwischen Werkstück und dem Handhabungsobjekt [9]. Mit anderen Worten: Die Kinematik beschreibt den mechanischen Aufbau des Roboters. Mithilfe von Werkzeugen oder Greifer wird die Wechselwirkung zwischen Roboter und Werkstück ermöglicht. Der Antrieb wandelt die notwendige Stellenergie um und überträgt sie zu Bewegungsachsen. Die Messsysteme erfassen die inneren Zustände wie Geschwindigkeit und Lage. Die Steuerung speichert, steuert und überwacht den gesamten Programmablauf. Über Sensoren werden Zustände des Handhabungsobjektes erfasst, physikalische Größen gemessen, Werkstücke identifiziert bzw. deren Lage ermittelt.

Im dreidimensionalen Raum ist die Lage eines starren Körpers durch seinen Ortsvektor und seine Orientierung durch eine Drehungsmatrix bestimmt [9]. Ein freibeweglicher Körper hat somit sechs Freiheitsgrade. Durch drei Verschiebungen und drei Drehungen kann ein Körper in jede beliebige Lage gebracht werden [9]. Dies entspricht der kombinierten Bewegung entlang der Translationsachsen und Rotationsachsen, wie sie beim Industrieroboter auftritt. Man unterscheidet daher bei der kinematischen Ausführung von Robotern zwei Grundbewegungen:

- translatorische Bewegungen und
- rotatorische Bewegungen.

Der mechanische Aufbau von Industrierobotern lässt eine Kombination dieser Bewegungen zu. Von einem fest verankerten Basiselement aus werden die Bewegungsachsen zu einer kinematischen Kette zusammengesetzt [9]. Die Achskombinationen können dabei variieren, bestimmen jedoch wesentlich Typ und Eigenschaft des Industrieroboters.

Außer Bewegungsform der Achsen kann man die Kinematik noch nach anderen Kriterien klassifizieren [9]:

- Anordnung der Achsen:

- Reihenfolge der Anordnung von rotatorischen und translatorischen Achsen
- Lage der Drehachsen von rotatorischen Achsen, Lage der Verschiebungsrichtung bei translatorischen Achsen

- Anzahl der Achsen
- Form des Arbeitsraumes
- Aus den drei oben genannten Kriterien und den Armlängen bzw. den Verfahrwegen ergibt sich die Form und Größe des Arbeitsraumes.
  - Folgende Formen sind gebräuchlich:
    - Kubus
    - Zylinder, Hohlzylinder
    - Kugel, Halbkugel, Hohlkugel

Aufgrund dieser Kriterien unterscheidet man verschiedene Typen von Industrierobotern

- Linearroboter
- Schwenkarmroboter (Scara-Roboter)
- Horizontal-Knickarmroboter
- Vertikal-Knickarmroboter

2.2.3 Charakteristische Eigenschaften von Industrierobotern

Die charakteristischen Eigenschaften von Industrierobotern sind im Detail in der deutschen Norm DIN EN 29946 (ISO 9946) „Industrieroboter: Darstellung charakteristischer Eigenschaften“ beschrieben. Die dort aufgelisteten Leistungskriterien sind genauer in der deutschen Fassung der EN 29283 „Leistungskriterien und zugehörige Testmethode“ erläutert.

Charakteristische Eigenschaften nach DIN EN 29946 sind:

- Anwendungsbereich
- Energieversorgung
- mechanische Struktur (Kinematik)
- Arbeitsraum

- Koordinatensysteme
- äußere Maße und Masse
- Basisaufstellfläche
- mechanische Schnittstelle
- Steuerung
- Programmierverfahren
- Umgebung
- Belastung
- Geschwindigkeit
- Auflösung
- Leistungskriterien
    - Pose-Genauigkeit in einer Richtung
    - Pose-Wiederholgenauigkeit in einer Richtung
    - Streuung der Pose-Genauigkeit in mehreren Richtungen
    - Abstandsgenauigkeit
    - Pose-Stabilisierungszeit
    - Pose-Überschwingen
    - Drift von Pose-Kenngrößen
    - Abweichungen beim Fahren einer Ecke
    - Kenngrößen der Bahngeschwindigkeit
    - Mindestpositionierzeit
    - Statische Nachgiebigkeit (Diese Leistungskriterien sind genauer in EN 29283 beschrieben. Diese europäische Norm ist bisher noch nicht in eine deutsche Norm überführt.)
- Sicherheit

2.2.4 Vergleich der Gelenkbauformen für Hauptachsen

1) Linearachsen
   - frei konfigurierbarer Arbeitsraum
   - beliebig erweiterbarer Arbeitsraum
   - günstige Kinematik für Handhabungs- und Palletieraufgaben
   - steife Gesamtkonstruktion durch mechanische Entkopplung der Achsen
   - große bewegte Massen

2) Rotatorische Achsen
   - schnelle Bewegung
   - kostengünstig für kleine Arbeitsräume
   - vorteilhafte Kinematik für Bearbeitungsaufgaben
   - unempfindlich gegen Verschmutzung
   - hoher Standarisierungsgrad
   - keine Linearitätsfehler

3) Zwangsgekoppelte Achsen
   - Kosteneinsparung
   - Reduzierung der Bewegungsfreiheiten
   - geringerer Energieverbrauch
   - höhere Sicherheit durch begrenzten Arbeitsraum

4) Parallelkinematiken
   - geringer Standarisierungsgrad (sehr teuer)
   - geringe bewegte Massen
   - sehr schnelle Bewegungen
   - kleiner Arbeitsraum
   - hohe Steifigkeit
   - aufwendige Steuerung

2.3 Prozessmodell

Für die Lösung einer Automatisierungsaufgabe muss ein Prozessmodell vorhanden sein. Das Prozessmodell ist ein mehr oder weniger abstraktes Abbild des realen Prozesses (Analogie). Es soll hinreichend genau das Verhalten des Prozesses unter der Wirkung der Einflussgrößen beschrieben. Eigenschaften des realen Prozesses, die für die Automatisierungsaufgabe nicht wichtig sind, werden weggelassen.

Das Prozessmodell beschreibt den Ausgang des Systems bei gegebenen Eingangsgrößen [7]. Es berücksichtigt dabei den inneren Zustand und das zeitliche Verhalten des zu beschreibenden Systems. Der Ausgang eines Prozessmodells ist sowohl vom Modellzustand als auch von den Eingangsgrößen abhängig. In vielen Fällen ist es auch notwendig, auftretende Störgrößen zu modellieren. Kleinere Abweichungen zwischen dem realen Prozess und dem Prozessmodell können so ausgeglichen werden.

Grundsätzlich unterscheidet man zwischen kontinuierlichen und diskreten Systemen. Sowohl kontinuierliche als auch ereignisdiskrete Systeme können zeitunabhängig (statisch) sein [7]. Ändert sich neben dem Zustand auch das Prozessmodell mit der Zeit, spricht man von dynamischen Systemen. Im Gegensatz zu diskreten Systemen sind kontinuierliche Systeme stets stetig. Abbildung 2.5 zeigt die Hierarchie des Prozessmodells.

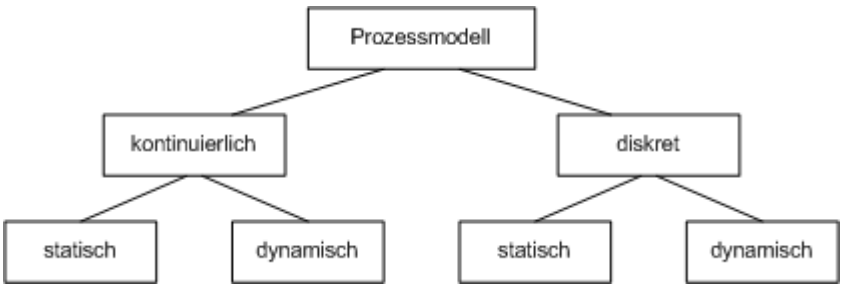


Abbildung 2.5 Kontinuierliche und diskrete Prozessmodelle [7]

Für kontinuierliche Prozessmodelle kommen häufig Differenzialgleichungen oder Vektordifferenzialgleichungen (im mehrdimensionalen Fall) zum Einsatz, die auf physikalischen Gesetzen basieren [7]. Diskrete Systeme hingegen werden oft mit Automaten oder Petrinetzen beschrieben. Der Weg, der zum jeweiligen Prozessmodell führt, wird als Modellierung bezeichnet und hat eine sehr große Bedeutung für den Entwurf von Automatisierungssystemen. Ist der zu modellierende reale Prozess hinreichend bekannt, so kann das Prozessmodell mathematisch in Form von Gleichungen und einem

Zustandsmodell formuliert werden. Liegt jedoch zu wenig Information über den zu modellierenden Prozess vor, muss eine Identifikation und Approximation des Prozesses durchgeführt werden [7].

2.4 Ereignisdiskrete Systeme

2.4.1 Ereignisse

Als Ereignis versteht man, dass es sowohl von außen (z.B. Benutzereingaben, Sensorwerte) als auch vom System selbst (z.B. Änderungsbenachrichtigungen) ausgelöst werden kann [10]. Zu beachten ist, dass man in der ereignisgesteuerten Architektur unter einem Ereignis immer die softwaretechnische Abbildung des tatsächlichen Ereignisses versteht und nicht das Ereignis selbst.

Ein Ereignis kann Auslöser (Trigger) für eine Ereignisbehandlung (Event Handling) sein, mit der das System reagiert. Eine ereignisgesteuerte Architektur hat wenig Kontrolle darüber, wann Daten verarbeitet werden [10]. Ein einfaches Beispiel ist die grafische Bedienoberfläche: Hier bestimmt der Benutzer, wann welche Daten verarbeitet werden, indem er Aktionen ausführt und damit Ereignisse auslöst.

Der Einsatz dieser Architektur setzt bei der Planung und Entwicklung voraus, dass alle Systeme, die an der Abwicklung des Ereignisses beteiligt sind, miteinander kommunizieren können [10]. Typischerweise erfordert die ereignisgesteuerte Architektur eine Definition dessen, was als Ereignis anzusehen ist. Dafür überwachen Computersysteme oder Sensoren den Status von Objekten und können ggf. ein Ereignis auslösen. Es folgen die Verarbeitung des Ereignisses nach definierten Regeln und die Folge des Ereignisses. Ist es während der Ereignisverarbeitung nicht möglich, sofort die definierte Folge des Ereignisses zu erzielen, wird das Ereignis im erreichten Status zwischengespeichert und erst weitergeführt, wenn die Folge erreicht werden kann.

Den größten Nutzen hat die ereignisgesteuerte Architektur, wenn sie bereits in der Planungsphase berücksichtigt wird. Bereits vorhandene Software auf eine ereignisgesteuerte Architektur umzustellen, kann mangels der benötigten Schnittstellen zu inakzeptablem Aufwand führen.

Die ereignisgesteuerte Architektur ergänzt die serviceorientierte Architektur, da Dienste durch Ereignisse ausgelöst werden können. Auf Basis der ereignisgesteuerten Archi-

tektur können insbesondere Systeme für ereignisgesteuerte Prozessketten entwickelt werden.

2.4.2 Ereignisfluss

Ein Ereignis umfasst meist drei Angaben: den Erstellungszeitpunkt (Zeitstempel), die auslösende Komponente (Quelle) und die Art des Ereignisses (Typ), die angibt, was im Wesentlichen vorgefallen ist [10]. Häufig wird der Ereignistyp nicht durch einen spezifischen Typ der jeweiligen Programmiersprache festgelegt, sondern durch hierarchisiertes Ereignisthema.

Ereignisse werden von Ereignisbehandlungsroutinen abgefangen, die bei Objektorientierung in Methoden oder auch eigenen Klassen implementiert werden. Ist eine Routine für einen abgefangenen Ereignistyp verantwortlich, löst sie passende Verarbeitungsschritte aus. Unabhängig davon kann die Routine das Ereignis an andere Routinen weiterreichen oder als erledigt markieren oder verwerfen.

Ereignisse sind keine Nachrichten, die zielgerichtet an bestimmte Komponenten versandt werden, sondern werden dem System gewissermaßen „blind“ übergeben [10]. Ein Ereignis kann an einer oder mehreren Stellen im System Aktionen auslösen, aber auch völlig unbeachtet verworfen werden (weil keine verarbeitende Routine als verantwortlich für das Ereignis gekennzeichnet wurde). Da Zeitpunkt und Reihenfolge, in der ein Ereignis verschiedene Komponenten erreicht, grundsätzlich zunächst nicht festgelegt sind, arbeiten die Routinen jeweils unabhängige, abgeschlossene Aufgaben ab. Ereignisgesteuerte Architektur ist daher prinzipiell auf Parallelverarbeitung ausgelegt.

2.4.3 Ereignisdiskrete Systeme

Die technische Entwicklung führt auf technische Systeme, die immer stärker von Computer geprägt und gesteuert werden. Diese Systeme sind durch eine asynchrone Arbeitsweise ihrer Teilsysteme geprägt, für deren Informationskopplungen diskrete Ereignisse eine dominante Rolle spielen. Durch diese Ereignisse löst eine Komponente eine bestimmte Aktivität in einer anderen Komponente aus, stoppt einen Vorgang oder beeinflusst dessen zukünftiges Verhalten. Ereignisse beschreiben dabei beispielsweise das Einschaltern eines Systems, die Ankunft eines Telefongesprächs, die Weiterleitung eines Datenpaketes oder die Grenzwertüberschreitung einer Temperatur. Darüber hinaus sind viele Entscheidungsregeln diskreter Natur. Sie beschreiben zum Beispiel Si-

tuationen, in denen Komponenten eines technischen Systems ein- oder ausgeschaltet oder Rechenprozesse gestartet oder gestoppt werden sollen.

Um das Verhalten derartiger Systeme zu verstehen und zu analysieren, muss man Modelle verwenden, die die asynchrone Arbeitsweise der Teilprozesse und den Informationsaustausch durch Ereignisse in den Mittelpunkt rücken [6]. Die hier verwendeten Betrachtungsweisen unterscheiden sich grundlegend von den in den Ingenieurwissenschaften seit langem eingesetzten Methoden, die die kontinuierlichen Verhaltensformen technischer und natürlicher Systeme in den Mittelpunkt stellen. Diese beschreiben Material-, Energie- und Informationsflüsse durch sich stetig ändernde, quantitative Größen wie Druck, Temperatur, Massenflüsse oder Datenraten, sodass die Modelle als Differenzialgleichungen gewinnt man aus den physikalischen Gesetzen, nach denen sich die betrachteten Prozesse abspielen.

Im Unterschied dazu ist das Verhalten von ereignisdiskreten Systemen durch Ereignisfolgen gekennzeichnet. Jedes Ereignis stellt eine plötzliche Änderung eines Eingangs-, Zustands- oder Ausgangssignals dar, wobei idealisierend angenommen wird, dass der Signalwechsel keine Zeit benötigt. Die Signale haben zwischen den abrupten Signalwechseln konstante Werte, die zu einer endlichen Menge diskreter Signalwerte gehören [6]. Der Signalverlauf lässt sich deshalb durch die Folge der nacheinander angenommenen Signalwerte darstellen. Diese Wertefolgen beschreiben zum Beispiel die zeitliche Änderung der Anzahl von Datenpaketen in einer Warteschlange, des Montagezustandes eines Werkstücks oder der Menge der bereits ausgeführten Rechenoperationen.

Dieses im Vergleich zu kontinuierlichen Systemen vollkommen andersartige Verhalten macht andersartige Modelle notwendig, die folglich auch eine andersartige mathematische Grundlage haben. Statt Differenzialgleichungen und algebraischen Gleichungen mit reellwertigen Signalen bilden Relationen zwischen diskreten Signalwerten die Grundform der Modelle [6]. Häufig kann man Signalverläufe nicht in analytischer Form darstellen, sondern muss die Folge von Signalwerten explizit aufschreiben. Grafentheoretische Verfahren sind die Basis für die Analyse des Systemverhaltens. Die Wahrscheinlichkeitstheorie wird verwendet, um bei nichtdeterministischen Zustandsübergängen die Häufigkeit zu beschreiben, mit der das System die einzelnen Zustandsübergänge durchläuft [6].

Der grundsätzliche Unterschied zwischen kontinuierlichen und ereignisdiskreten Systemen liegt im Wertebereich der Signale. Während kontinuierliche Signale einen reell-

wertigen Wertebereich mit unendlich vielen möglichen Signalwerten haben, besitzen diskrete Signale einen Wertebereich mit endlich vielen einzelnen Werten oder zumindest abzählbar unendlich vielen Werten [6].

Beispiel: Ereignisdiskrete Arbeitsweise eines Getränkeautomaten

Die Kaffeezubereitung in einem Getränkeautomaten umfasst eine Reihe von Teilprozessen:

- Becher unter den Auslauf stellen
- Getränkeauswahl treffen
- Kaffee abmessen
- Kaffee mahlen
- Wasser zum Kochen bringen
- Kaffeepulver in einen Filter füllen
- Wasser durch den Kaffeefilter in den Becher laufen lassen

Diese Vorgänge laufen sequenziell oder parallel ab. Beispielsweise muss der Becher unter den Auslauf gestellt werden, bevor das kochende Wasser durch den Filter fließt. Anderseits kann das Wasser zum Kochen gebracht und gleichzeitig in den Filter geleitet werden, sodass diese beiden Teilprozesse parallel ablaufen.

Um den Getränkeautomaten so zu steuern, dass die Teilprozesse in der gewünschten Reihenfolge und in der erforderlichen Dauer ablaufen, muss man die Teilprozesse zum richtigen Zeitpunkt starten und beenden. Das Ein- und Ausschalten der entsprechenden Komponenten des Getränkeautomaten sind Ereignisse, von denen jedes im Vergleich zu den beschriebenen Teilprozessen in vernachlässigbar kurzer Zeit abläuft. Das Verhalten des Getränkeautomaten wird durch die Folge dieser Ereignisse beschrieben.

Der Getränkeautomat erfüllt seine Funktionen, wenn die Ereignisse in der gewünschten Reihenfolge auftreten. Funktionsstörungen können deshalb daran erkannt werden, dass gewünschte Ereignisse nicht vorkommen, zusätzliche Ereignisse auftreten, oder die Ereignisse die falsche Reihenfolge haben. Wenn man die Funktionsweise des Getränkeautomaten verstehen bzw. kontrollieren will, ist die Verhaltensbeschreibung durch eine Folge von Ergebnisse reagiert und Steuerereignisse vorgibt. Beispielsweise meldet die Kaffeemühle der Steuerung durch ein binäres Signal, dass die abgemesse-

ne Menge Kaffe vollständig gemahlen ist. Dies zeigt ein Ereignis an, nach dem die Steuerung die Heizung einschaltet.

Die Abbildung 2.6 beschreibt das Modell der Steuerung des Getränkeautomaten.

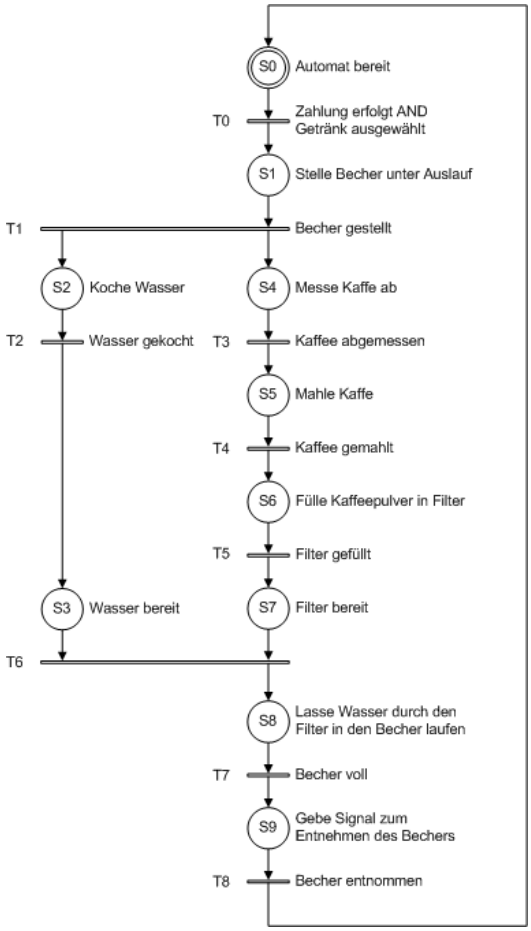


Abbildung 2.6 Modell der Steuerung des Getränkeautomaten

Gründe für die ereignisdiskrete Betrachtungsweise: Ob ein System durch ein kontinuierliches oder ein ereignisdiskretes Modell beschrieben wird, hängt nicht nur vom Charakter der Prozesse ab, die in dem System ablaufen, sondern auch vom Modellbildungsziel ab [6]. Es ist in vielen Situationen zweckmäßig, von kontinuierlichen Signalverläufen zu abstrahieren und Signale diskret zu beschreiben. Dieser Situation entsprechend gibt es zwei Gründe für eine ereignisdiskrete Betrachtung dynamischer Systeme:

1) Die Signale haben aus physikalischen oder technischen Gründen einen diskreten Wertebereich [6].

In dieser Situation hat man bei der Modellierung keine andere Wahl, als mit diskreten Signalen zu arbeiten. So springen die internen Signale informationsverarbeitender Systeme in Abhängigkeit von der eingelesenen Zeichenkette zwischen diskreten Werten hin und her und erzeugen eine wertdiskrete Ausgabefolge. Die ereignisdiskrete Betrachtungsweise ist deshalb in der Informatik sowie in wichtigen Gebieten der Informationstechnik weit verbreitet.

Auch beim Schalten eines Getriebes oder eines Schalters gibt es keine Zwischenwerte und man beschreibt den aktuellen Zustand durch eine ganzzahlige Größe. Bei der Paketvermittlung im Internet wird der Netzzustand durch das Vorhandensein oder Nichtvorhandensein ganzer Datenpakete beschrieben. In digitalen Schaltungen wird aus Gründen der Robustheit nur mit zwei Signalpegeln gearbeitet und durch die Schaltungstechnik erreicht, dass die kontinuierlichen Übergänge zwischen diesen Signalpegeln in so kurzer Zeit ablaufen, dass sie die Funktionsweise der Schaltung nicht maßgebend sind. Wenn ein System über Schaltventile gesteuert und das Verhalten über Endlagenschalter beobachtet wird, sind für das Modell ebenfalls nur diskrete Signalwerte maßgebend, obwohl unterlagerte physikalische Vorgänge durchaus in anderen Zusammenhängen kontinuierlich modelliert werden könnten.

2) Diskrete Signale entstehen durch Abstraktion aus kontinuierlichen Signalen [6].

In diese Situation werden bei der Modellierung nur diskrete Beschreibungen genutzt, obwohl die Signale reellwertig sind. Durch die Abstraktion werden die möglichen Zustände des Systems auf eine endliche Menge beschränkt. Damit verbunden ist die Darstellung kontinuierlicher Signaländerungen als Schalten des Signals von einem diskreten Wert zum nächsten. Dabei stellt sich der Nachfolgewert des Signals ohne Zeitverzögerung nach dem alten Signalwert ein.

3 Problem- und Aufgabenstellung

3.1 Problemstellung

Viele Hersteller im Robotik-Bereich produzieren nicht nur Roboter, sondern haben eine selbstentwickelte Steuerungssoftware für ihre Maschinen. Falls ein Roboter-Benutzer die Steuerungssoftware des Herstellers verwenden muss, entstehen in einigen Fällen Schwierigkeiten, zum Beispiel:

- Die Kosten der Software sind zu hoch.
- Die Software ist zu kompliziert zu verwenden.
- Man kann keine neuen Funktionen erstellen, damit der Roboter intelligenter wird oder andere Aufgaben erfüllen kann usw.

Anders als früher werden in der heutigen Zeit immer mehr Produktionsbereiche in viele Unterteile spezialisiert. Als Beispiel: Ein Autohersteller produziert nicht alle Komponenten eines Wagens von A bis Z, sondern er kauft die Reifen von einer Firma, die Steuerungssoftware für das Auto von einer anderen usw. Diese Spezialisierung hat den Vorteil, dass sich jedes Unternehmen auf sein wesentliches Fachgebiet konzentrieren kann und seine Produkte dadurch in höherer Qualität hergestellt werden.

Deshalb stellt sich die Frage: Warum entwickelt eine Firma nicht eine Steuerung, die mit unterschiedlichen Roboter funktioniert? Je nach Art eines Roboters oder einer Aufgabe können unterschiedliche Funktionen erstellt werden. Darüber hinaus wird die Steuerung nicht nur im Robotik-Bereich eingeschränkt, sondern wird in anderen Bereichen, zum Beispiel Anlagensteuerung oder Produktion, eingesetzt.

3.2 Aufgabenstellung

Im Rahmen dieser Diplomarbeit soll die oben vorgestellte Idee in einem Robotik-Experiment untersucht, analysiert und beschrieben werden. Die Arbeit wird von der Firma IFS Informationstechnik unterstützt. Die Steuerung Heron, die für die Robotersteuerung verwendet wird, wurde von dieser Firma entwickelt. Der Roboter ist ein Produkt des Herstellers Fischertechnik. Die Hauptpunkte der Diplomarbeit sind folgende:

- Untersuchung des mechanischen Aufbaus der Sensoren/Aktoren und der Grundfunktionen des Roboters von Fischertechnik
- Entwicklung intelligenter Sensoren/Aktoren, um neue Funktionen für den Roboter zu realisieren
- Definition einer Transportaufgabe, die vom Roboter gelöst werden soll: Durch diese Aufgabe sollen alle Funktionen des Roboters sowie die Kommunikation zwischen dem Roboter und der Steuerung getestet werden.
- Entwurf der Steuerung als zeitbehaftetes, interpretiertes Petrinetz und Darstellung in der kanonischen Beschreibungsform (KBF) (Die ausführliche Beschreibung der KBF wird in Kapitel 4.2 vorgestellt).
- Test der Steuerung am Roboter; Erfassen typischer Kenngrößen der Steuerungstechnik (z.B. Reaktionszeiten)
- Erstellen eines Technischen Berichts

3.3 Systemgrenze

Nach der Aufgabenstellung sollen ein System und eine entsprechende Systemgrenze definiert werden. Das Untersuchungssystem besteht aus den folgenden Komponenten:

- Steuerung
- Automatenmodell
- Bibliotheksdateien mit Bibliotheksfunktionen
- Roboter und dazugehörender Treiber

Diese Systemobjekte stehen miteinander in Beziehung und wirken aufeinander ein. Fehlt eine dieser Komponenten, kann das System nicht funktionieren. Außerdem wer-

den sie durch die Systemgrenze mit ihrer Umwelt abgegrenzt. Ein Element des Systems kann ein Element der Umwelt bewirken und umgekehrt. Ihre Funktionsfähigkeiten sind aber nicht voneinander abhängig. Als Beispiel: Der Roboter kann ein Werkstück aufnehmen und transportieren. Ohne dieses Werkstück funktioniert das System trotzdem, aber nicht ohne die Steuerung oder das Automatenmodell. Die Abbildung 3.1 stellt das Untersuchungssystem, dessen Systemgrenze und die Beziehung zwischen den Elementen dar.

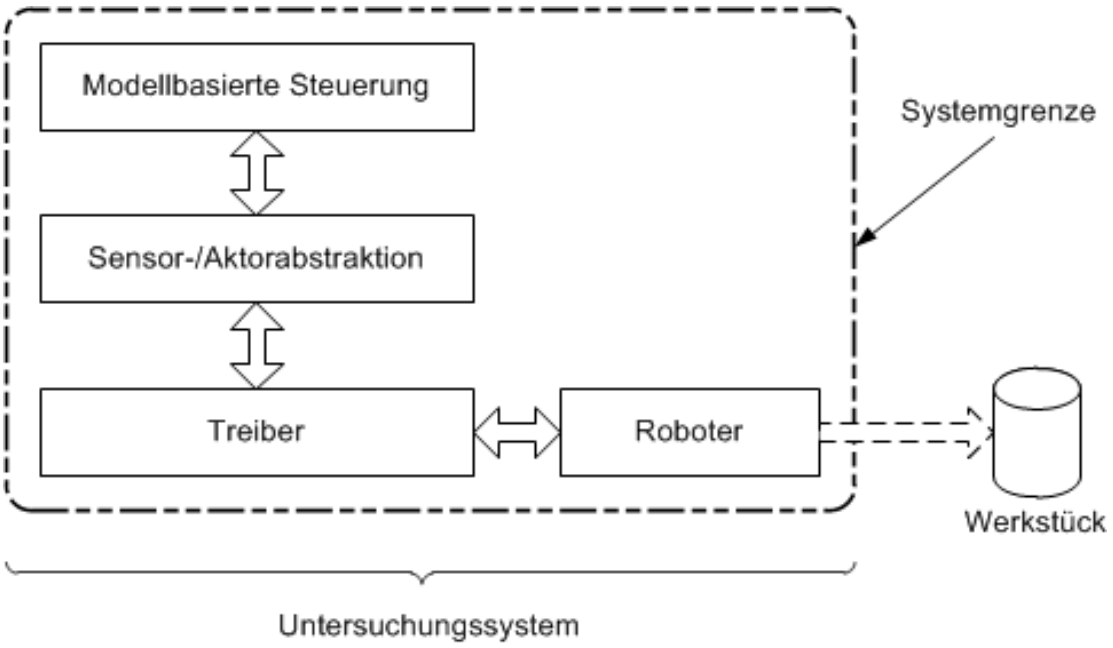


Abbildung 3.1 Untersuchungssystem und Systemgrenze

4 Software-Komponenten des Untersuchungssystems

4.1 Anwendungen

4.1.1 Heron

Heron ist eine echtzeitfähige Steuerung, die von der Firma IFS Informationstechnik entwickelt wurde. Heron kann Petrinetze, die in der kanonischen Beschreibungsform (KBF) vorliegen, übersetzen und ausführen. Die wichtigsten Anforderungen an Heron waren Echtzeitfähigkeit und Robustheit.

4.1.1.1 Einleitung zu Sensoren/Aktoren

Sensoren werden auch als Messgrößenaufnehmer, Messfühler oder Detektoren gezeichnet. Sie sind technische Bauteile, die physikalische oder chemische Eigenschaften wahrnehmen und messen können [12]. Die von Sensoren erfassten Messwerte werden in Größen umgeformt, die weiterverarbeitet werden können (meistens in elektrische Signale).

Aktoren sind elektrische Einrichtungen, die Informationen empfangen und die zugehörigen Aktionen ausführen [3]. Im Rahmen einer Steuerung bilden sie die Gegenstücke zu den Sensoren, indem sie elektrische Signale in nichtelektrische Aktionen umsetzen.

In dem Untersuchungssystem dieser Arbeit beschreiben die Begriffe Sensoren/Aktoren auch die softwaretechnischen Funktionen, die im Prinzip der Sensoren/Aktoren arbeiten. Diese Funktionen werden daneben als Sensoren-/Aktoren-Methoden gezeichnet. Die Sensoren-Methoden können Signale oder Messwerte erfassen und zurückgeben. Die Aktoren-Methoden können ihre Umgebung bewirken wie zum Beispiel: einen Steuerbefehl eines Motors an die Steuerung senden, Verbindung zwischen einer Steuerung und einem Roboter aufbauen usw. Alle interne Sensoren/Aktoren von Heron sind Sensoren-/Aktoren-Methoden.

4.1.1.2 Interne Sensoren und Aktoren

In Heron existieren zwei interne Module mit Sensoren und Aktoren: *TIMER* und *MEMORY*. Alle internen Sensoren sind aktiv implementiert. Wenn externe Module mit dem gleichen Namen verwendet werden, werden die internen Module durch diese externen Module ersetzt.

Modul *TIMER*

Das Modul *TIMER* stellt Zeitgeber zur Verfügung. Es können beliebig viele Zeitgeber gleichzeitig verwendet werden. Die einzelnen Zeitgeber werden über einen eindeutigen Namen identifiziert, der allen Methoden als Parameter übergeben wird. Tabelle 4.1 beschreibt die Funktion der Methoden des Moduls *TIMER*.

Methode	Beschreibung
IsExpired	Diese Sensoren-Methode prüft, ob ein Zeitgeber abgelaufen ist. Der einzige Parameter ist der Name (*name*) des Zeitgebers.
StartTimer	Aktoren-Methode zum Starten eines Zeitgebers; Parameter sind der Name (*name*) des Zeitgebers, der Typ (*type*: *Singular* oder *Periodic*) und die Dauer, die über die Parameter *duration* und *unit* (ms, sec, min, h oder day) angegeben wird.
StopTimer	Aktoren-Methode zum Beenden eines Zeitgebers. Der einzige Parameter ist der Name (*name*) des Zeitgebers.

Tabelle 4.1 Methoden des Moduls *TIMER*

Modul *MEMORY*

Das Modul *MEMORY* dient der Speicherung von Daten. Die verschiedenen Speicherstellen werden über einen eindeutigen Namen identifiziert, der in jeder Methode übergeben wird.

Methode	Beschreibung
GetValue	Diese Sensoren-Methode ruft ein gespeichertes Objekt ab. Der einzige Parameter ist der Name (*name*) der Speicherstelle. Wurde noch kein Objekt mit diesem Namen gespeichert, gibt die Methode null zurück.
HasValue	Diese Sensoren-Methode prüft, ob eine bestimmte Spei-cherstelle angelegt wurde. Der einzige Parameter ist der Name (*name*) der Speicherstelle.

SetValue	Aktoren-Methode zum Speichern eines Objekts. Parameter sind der Name (*name*) der Speicherstelle und das zu speichernde Objekt (*value*). Existiert die Speicherstelle bereits, wird das vorherige Objekt überschrieben. Durch Schreiben des Werts null wird die Speicherstelle wieder gelöscht.

Tabelle 4.2 Methoden des Moduls *MEMORY*

Die beiden folgenden Aktoren-Methoden können nur verwendet werden, wenn in der Speicherstelle ein numerischer Wert gespeichert ist.

Methode	Beschreibung
Increase	Aktoren-Methode zum Erhöhen des gespeicherten Werts um 1. Der einzige Parameter ist der Name (*counter*) der Speicherstelle.
Decrease	Aktoren-Methode zum Vermindern des gespeicherten Werts um 1. Der einzige Parameter ist der Name (*counter*) der Speicherstelle.

Tabelle 4.3 Weitere Methoden des Moduls *MEMORY*

4.1.1.3 Externe Sensoren und Aktoren

Außer den internen Sensoren und Aktoren, die oben beschrieben werden, kann Heron auch externe Sensoren und Aktoren anbinden. Für jeden Zweck und jede Hardware benötigt man eventuell verschiedene Sensoren und Aktoren. Zum Beispiel: Für die Steuerung eines Roboters braucht man die Sensoren-Methoden, um die aktuelle Position zu erkennen oder die Aktoren-Methoden, um die Motoren zu steuern. Man kann nicht alle brauchbaren Sensoren und Aktoren in eine Steuerung einbauen. Die werden in externe Module implementiert. Wenn die benötigt werden, bindet man die Module an die Steuerung an.

Externe Module für Sensoren und Aktoren können auf zwei Arten in Heron bekannt gemacht werden. In der Regel werden sie in der Konfigurationsdatei *HeronConfiguration.xml* eingetragen. In diesem Fall lädt Heron die Module selbst und erzeugt Instanzen der benötigten Sensoren und Aktoren. Die andere Möglichkeit ist, dass eine externe Anwendung über die Schnittstellen-Methode *IHeron.RegisterSensorActorModule* bestehende Instanzen von Sensoren und Aktoren bei Heron registriert. Auf diese Weise kann eine externe Anwendung eine Verbindung zu Heron herstellen.

4.1.2 Auriga

Auriga ist eine Anwendung, die von der Firma IFS Informationstechnik entwickelt wurde. Diese Software kann ein Automatenmodell, das in der kanonischen Beschreibungsform (siehe Kapitel 4.2) erzeugt ist, als GRAFCET (siehe Kapitel 4.3) darstellen. Durch eine Schnittstelle verbindet sich Auriga mit Heron. Während Heron ein Modell ausführt, stellt Auriga parallel die gerade aktiven Zustände des Modells in Echtzeit dar. Mit dieser Fähigkeit von Auriga kann man Zustände eines laufenden Steuerungsprozesses betrachten und nachvollziehen. Darüber hinaus vereinfacht Auriga die Arbeit, Fehler oder Ausnahmen zu erkennen. Ein Bildschirmfoto der Anwendungsoberfläche von Auriga als Beispiel wird in der Abbildung 4.1 dargestellt.

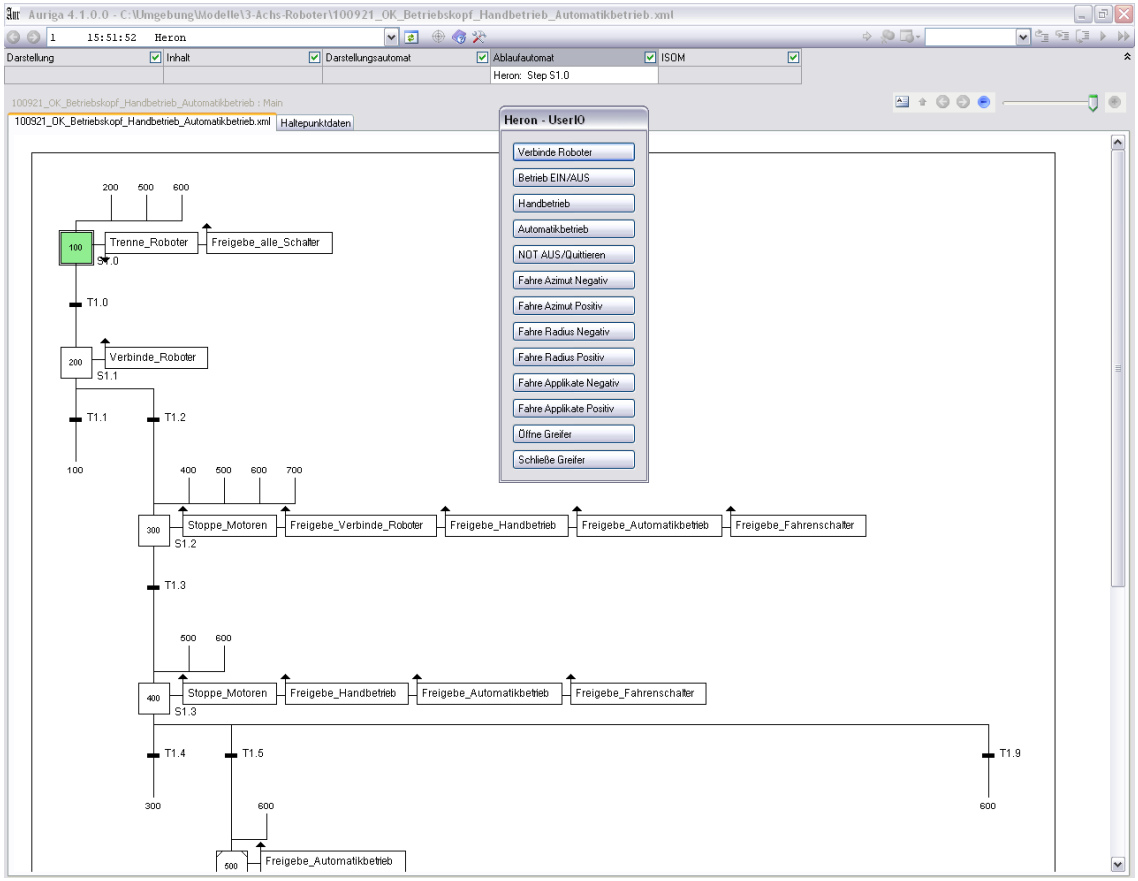


Abbildung 4.1 Anwendungsoberfläche von Auriga

4.2 Kanonische Beschreibungsform (KBF)

Es gibt verschiedene technische Beschreibungsformen für Automatenmodelle. Eine davon ist die kanonische Beschreibungsform (KBF), die von der Firma IFS Informationstechnik entwickelt wurde. Dabei handelt es sich um ein XML-Format. Eine der wichtigsten Anforderungen an die kanonische Beschreibungsform ist die Fähigkeit, Nebenläufigkeiten darstellen zu können.

Ein XML-Dokument, das ein Automatenmodell in KBF beschreibt, wird in mehrere Abschnitte gegliedert. Diese Abschnitte werden nach Elemente der KBF dargestellt. Eine KBF haben sieben Basiselemente:

- Sensoren (als *Sensor* in KBF gekennzeichnet)
- Ereignisse (*Event*)
- Aktoren (*Actor*)
- Aktionen (*Action*)
- Bedingungen (*Condition*)
- Zustände/Übergeordnete Zustände (*State*/*SuperState*)
- Transitionen (*Transition*)

Dabei werden Sensoren und Ereignisse als Eingänge (*Inputs*) und Aktoren und Aktionen als Ausgänge (*Outputs*) zusammengefasst.

Um den Aufbau der KBF und die Struktur jedes Elements zu veranschaulichen, wird ein gekürzter Auszug aus einem der Beispiele von Heron verwendet (siehe Codeauszug 4.1). Das Beispiel beschreit die Steuerung einer Ampel-Lampe.

```
<?xml version="1.0" encoding="utf-8"?>
<KBF xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="KBF.xsd">
  <!-- ========================================================== -->
  <Inputs>
    <Sensors>
      <Sensor id ="Timer_Rot_abgelaufen" description="Zeitgeber für die Rotphase"
      module="TIMER“ operation="IsExpired">
        <Parameter name="name" value="Timer_Rot"/>
      </Sensor>
      …
    </Sensors>
    <!-- ========================================================== -->
```

```
  <Events>
    <Event id="Rot_vorbei" description="Rotphase beenden">
      <Sensor ref="Timer_Rot_abgelaufen" type="rising"/>
    </Event>
    …
  </Events>
</Inputs>
<!-- ============================================================ -->
<Outputs>
  <Actors>
    <Actor id ="Uhr_aufziehen_Dauer_Rot" description="Uhr für Rot-phase aufziehen"
    module="TIMER" operation="StartTimer">
      <Parameter name="name" value="Timer_Rot"/>
      <Parameter name="Type" value="Singular"/>
      <Parameter name="Duration" value="5"/>
      <Parameter name="Unit" value="sec"/>
    </Actor>
    …
  </Actors>
  <!-- ============================================================ -->
  <Actions>
    <Action id="Uhr_Rot" description="Starte Zeitgeber fuer Rotphase">
      <Actor ref="Uhr_aufziehen_Dauer_Rot" trigger="Activate"   result="Erfolg" />
    </Action>
  …
  </Actions>
</Outputs>
<!-- ============================================================ -->
<Conditions>
  <Condition id="C11" description="Bedingung fuer Transition Rotphase zu
  Rot/Gelbphase">
    Rot_vorbei
  </Condition>
  …
</Conditions>
<!-- ============================================================ -->
<States>
  <State id="S1" description="Fahrzeug-Ampel rot" initial="1">
    <Actions>
      <Action ref="Fahrzeugampel_Rotlicht"/>
    </Actions>
    <Enables>
      <Transition ref="T13" />
      <Transition ref="T11" />
    </Enables>
  </State>
  …
  <SuperState id="Betrieb" description="Ampel arbeitet normal">
```

```
      <Actions></Actions>
      <Enables>
        <Transition ref="T_ERROR"/>
      </Enables>
      <ContainedStates>
        <State ref="S1"/>
        <State ref="S2"/>
        <State ref="S3"/>
      </ContainedStates>
      <Activates>
        <State ref="S1"/>
      </Activates>
    </SuperState>
    …
  </States>
  <!-- ============================================================ -->
  <Transitions>
    <Transition id="T21" description="Gelbphase zu Rotphase" condition="C21" >
      <Actions>
        <Action ref="Uhr_Rot" />
      </Actions>
      <NextStates>
        <State ref="S1" />
      </NextStates>
    </Transition>
    …
  </Transitions>
  <!-- ============================================================ -->
  <Sources>
  <Sources />
  <Hints>
    <Hint name="ValidationMode" value="tolerant" />
  </Hints>
</KBF>
```

Codeauszug 4.1 Gekürzter Auszug der Steuerung einer Ampel-Lampe in KBF

4.2.1 Sensoren (*Sensors*)

Sensoren erfassen eingehende Signale. Jeder Eintrag eines Sensors definiert einen Methodenaufruf, der den betreffenden Messwert ermittelt (siehe Codeauszug 4.2). Die Zuordnung der „Module“ zu ausführbaren Dateien erfolgt über die Konfigurationsdatei der Steuerung Heron.

```
<Sensor id ="Timer_Rot_abgelaufen" description="Zeitgeber für die Rotphase"
module="TIMER" operation="IsExpired">
```

```
  <Parameter name="name" value="Timer_Rot"/>
</Sensor>
```

Codeauszug 4.2 Beispiel für Sensor

Zwingend erforderlich sind ein eindeutiger Bezeichner des Sensors (*Id*), der Name des Moduls (*module*) und der Sensoren-Methode (*operation*). Optional kann eine Beschreibung (*description*) angegeben werden. Anzahl und Namen der Parameter müssen zu der Parameterliste der Sensoren-Methode passen.

4.2.2 Ereignisse (*Events*)

Ereignisse bewerten die Messwerte der Sensoren. Durch Schwellwerte (*threshold*) oder Vergleichswerte (*match*) werden die Messwerte eingeordnet. Wenn der Messwert bereits an Wahrheitswert (Boolean) ist, entfällt die Angabe von Schwellwert oder Vergleichswert. Der Typ des Sensors legt fest, wann eine Änderung des Messwertes das Ereignis signalisiert. Codeauszug 4.3 zeigt einen Abschnitt in KBF als Beispiel für ein Ereignis.

```
<Event id="Rot_vorbei" description="Rotphase beenden">
  <Sensor ref="Timer_Rot_abgelaufen" type="rising"/>
</Event>
```

Codeauszug 4.3 Beispiel für Ereignis

Zwingend erforderlich ist ein eindeutiger Bezeichner (*id*), der Verweis (*ref*) auf einen Sensor und der Abfragetyp des Sensorwerts (*type*) (siehe Tabelle 4.4). Der Sensor muss auf einen existierenden Sensor-Eintrag verweisen. Optional kann eine Beschreibung (*description*) angegeben werden.

Typ	Beschreibung
high bzw. true	Signal ist über der Schwelle, gleich dem Vergleichswert oder *true*
low bzw. false	Signal ist unter der Schwelle, ungleich dem Vergleichswert oder *false*
rising	Wechsel von *low* nach *high*
falling	Wechsel von *high* nach *low*

Tabelle 4.4 Abfragetype des Sensorwerts

Die Ergebnisse werden in den Transitionsbedingungen verwendet.

4.2.3 Aktoren (*Actors*)

Aktoren sind das Gegenstück zu den Sensoren. Jeder Eintrag eines Aktors definiert einen Methodenaufruf, der die betreffende Wirkung nach außen veranlasst (siehe Codeauszug 4.4). Die Zuordnung der Module zu ausführbaren Dateien erfolgt über die Konfigurationsdatei der Steuerung Heron.

```
<Actor id ="Uhr_aufziehen_Dauer_Rot" description="Uhr für Rot-phase aufziehen"
module="TIMER" operation="StartTimer">
  <Parameter name="name" value="Timer_Rot"/>
  <Parameter name="Type" value="Singular"/>
  <Parameter name="Duration" value="5"/>
  <Parameter name="Unit" value="sec"/>
</Actor>
```

Codeauszug 4.4 Beispiel für Aktor

Zwingend erforderlich sind ein eindeutiger Bezeichner (*Id*), der Name des Moduls (*module*) und der Aktoren-Methoden (*operation*). Optional kann eine Beschreibung (*description*) angegeben werden. Anzahl und Namen der Parameter müssen zu der Parameterliste der Aktoren-Methode passen.

4.2.4 Aktionen (*Actions*)

Aktionen referenzieren einen oder mehrere Aktoren. Sie legen fest, wann die Aktoren aufgerufen werden (siehe Codeauszug 4.5). Außerdem besteht die Möglichkeit, das Ergebnis eines Aktors zu speichern. Dazu vergibt man über result einen eindeutigen Namen für das Ergebnis, das mithilfe der Methode GetResult des internen Moduls *MEMORY* abgefragt werden kann. Dabei ist zu beachten, dass Ergebnisse von Aktionen nur in den Bedingungen der unmittelbar folgenden Transitionen verarbeitet werden können.

```
<Action id=" Uhr_Rot" description="Starte Zeitgeber für Rotphase">
  <Actor ref="Uhr_aufziehen_Dauer_Rot" trigger="Activate"
  result="Erfolg" />
</Action>
```

Codeauszug 4.5 Beispiel für Aktion

Zwingend erforderlich sind ein eindeutiger Bezeichner (*Id*) und der Verweis auf einen Aktor. Optional kann eine Beschreibung (*description*) angegeben werden. Die Aktoren müssen auf existierende Aktor-Einträge verweisen.

Durch die Angabe von jeweils einem Aktor mit *trigger="Activate"* und *trigger="Deactivate"* kann eine einzelne Aktion definiert werden, die beim Aktivieren oder Deaktivieren einer Stelle die entsprechenden Aktoren betätigt. Bei Aktionen an Transitionen wird die Angabe eines *trigger* ignoriert.

4.2.5 Bedingungen (*Conditions*)

Basierend auf den zuvor definierten Ereignissen werden hier Transitionsbedingungen festgelegt (siehe Codeauszug 4.6). Neben den Namen der zuvor definierten Ereignisse können in den Bedingungen die Namen von Stellen und die Schlüsselwörter *AND* und *NOT* verwendet werden. Stellen sind *true*, wenn sie aktiv sind. Die Schlüsselwörter *true* und *false* können für invariante Terme verwendet werden. Leere Bedingungen sind immer *true*.

```
<Condition id="C11" description="Bedingung fuer Transition Rotphase zu Rot/Gelbphase">
   Rot_vorbei
</Condition>
```

Codeauszug 4.6 Beispiel für Bedingung

Zwingend erforderlich ist ein eindeutiger Bezeichner (*Id*). Optional kann eine Beschreibung (*description*) angegeben werden. Der Bedingungsterm darf nur die oben genannten Schlüsselwörter und die Namen von im Modell existierenden Ereignissen und Stellen enthalten. Wenn ein Ereignis und eine Stelle mit der gleichen *Id* im Modell existieren, gilt der Verweis im Term dem Ereignis.

4.2.6 Zustände (*States*)

Hier werden alle Stellen bzw. Zustände des Modells aufgeführt. Jede Stelle kann beliebige zuvor definierte Aktionen referenzieren. Außerdem muss jede Stelle angeben, welche Transitionen von ihr aus erreichbar sind (siehe Codeauszug 4.7).

```
<State id="S1" description="Fahrzeug-Ampel rot" initial="1">
  <Actions>
    <Action ref="Fahrzeugampel_Rotlicht"/>
  </Actions>
  <Enables>
    <Transition ref="T13" />
    <Transition ref="T11" />
  </Enables>
</State>
```

Codeauszug 4.7 Beispiel für Zustand

Zwingend erforderlich ist ein eindeutiger Bezeichner (*Id*). Optional kann eine Beschreibung (*description*) angegeben werden.

Über die Eigenschaft *initial* wird festgelegt, wie viele Marken die Stelle zu Beginn haben soll. Der Vorgabewert ist 0.

4.2.7 Übergeordnete Zustände (*SuperStates*)

Ein übergeordneter Zustand erweitert einen normalen Zustand um die Listen *ContainedStates* und *Activates* (siehe Codeauszug 4.8). Die Einträge der Liste *ContainedStates* verweisen auf andere Zustände, die in dem übergeordneten Zustand eingeschlossen sind. Solange der übergeordnete Zustand aktiv ist, können auch die eingeschlossenen Zustände aktiviert werden. Wird der übergeordnete Zustand deaktiviert, werden auch alle Marken aus den eingeschlossenen Zuständen abgezogen.

Die Einträge der Liste *Activates* müssen eine Teilmenge der Einträge in *ContainedStates* sein. Die hier aufgeführten Zustände werden aktiviert, wenn der übergeordnete Zustand aktiviert wird.

Ein typischer Fall für die Verwendung eines übergeordneten Zustandes ist die Fehlerbehandlung. Ohne übergeordneten Zustand müsste jeder Zustand des Automaten Transitionen für alle möglichen Fehlerfälle enthalten. Mit übergeordnetem Zustand werden die Transitionen der Fehlerfälle nur am übergeordneten Zustand angegeben und alle relevanten Zustände in dem übergeordneten Zustand eingeschlossen.

```
<SuperState id="Betrieb" description="Ampel arbeitet normal">
  <Actions></Actions>
  <Enables>
    <Transition ref="T_ERROR"/>
  </Enables>
  <ContainedStates>
    <State ref="S1"/>
    <State ref="S2"/>
    <State ref="S3"/>
  </ContainedStates>
  <Activates>
    <State ref="S1"/>
  </Activates>
</SuperState>
```

Codeauszug 4.8 Beispiel für übergeordneten Zustand

4.2.8 Transitionen (*Transitions*)

Hier werden alle Transitionen des Modells aufgeführt. Der Codeauszug 4.9 beschreibt die Struktur einer Transition in KBF.

```
<Transition id="T21" description="Gelbphase zu Rotphase" condition="C21">
  <Actions>
    <Action ref ="Uhr_Rot" />
  </Actions>
  <NextStates>
    <State ref="S1" />
  </NextStates>
</Transition>
```

Codeauszug 4.9 Beispiel für Transition

Zwingend erforderlich sind ein eindeutiger Bezeichnung (*Id*) und der Verweis auf eine Schaltbedingung (*condition*). Optional kann eine Beschreibung (*description*) angegeben werden. Jede Transition kann beliebige zuvor definierte Aktionen referenzieren. Außerdem muss die Liste mit Verweisen auf Folgezustände angegeben werden.

4.2.9 Zusammenhang zwischen den Elementen einer KBF

Nachdem alle Elementen einer KBF ausführlich beschrieben wurden, sollen die Zusammenhänge zwischen diesen dargestellt werden, um den Aufbau der KBF noch veranschaulichter zu machen. Die Abbildung 4.2 stellt die Struktur einer KBF und die Zusammenhänge zwischen den Elementen dar.

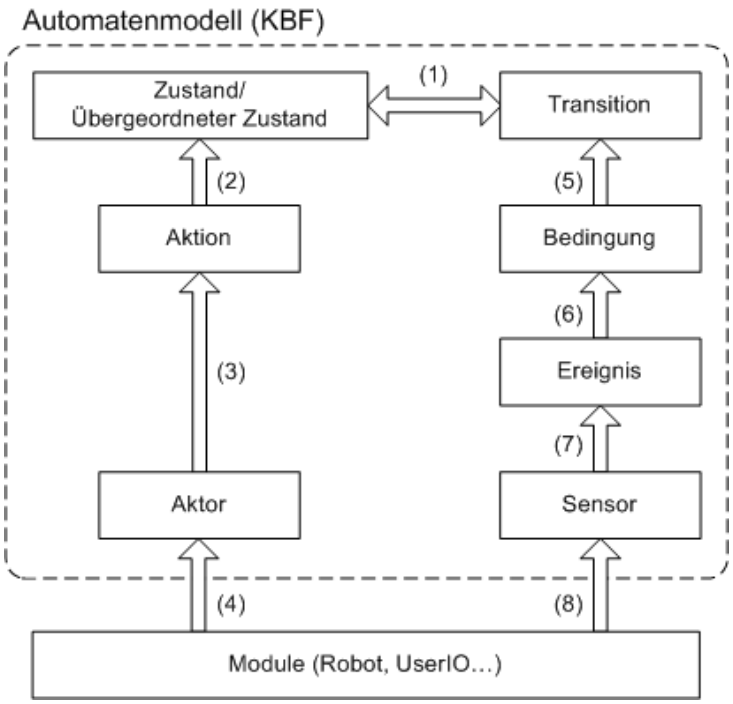


Abbildung 4.2 Struktur einer KBF und Zusammenhänge zwischen Elementen

1) In KBF führen Zustände zu Transitionen hin und umgekehrt.
2) Ein aktiver Zustand kann eine oder mehrere Aktionen durchführen.
3) Eine Aktion kann das Ergebnis der Ausführung von einem oder vielen Aktoren sein.
4) Aktoren werden von Modulen aufgerufen.
5) Eine Transition hat eine Schaltbedingung.
6) Eine Bedingung kann mehrere Ereignisse erhalten.
7) Ereignisse werden durch die Messwerte von Sensoren hergestellt.
8) Die Sensoren werden von Modulen abgerufen.

4.3 GRAFCET

GRAFCET ist eine Spezifikationssprache, die vor allem in der Automatisierungs- und Verfahrenstechnik für die Darstellung von Ablaufbeschreibungen verwendet wird. GRAFCET ist eine Abkürzung, die aus dem französischen stammt: GRAphe Fonctionnel de Commande Etape Transitions [1]. Das bedeutet auf Deutsch: Darstellung der

Steuerungsfunktion mit Schritten und Weiterschaltbedingungen. GRAFCET beschreibt das Verhalten der Steuerung unabhängig von der technischen Realisierung. Mögliche Abläufe zwischen den Zuständen der Steuerung werden durch die Struktur einer GRAFCET-Abbildung gezeigt. Die Sprache ist in der Norm zur DIN EN 60848 beschrieben. Als Nachfolger von DIN 40719-Teil 6 „Funktionsplan“ besitzt GRAFCET im Gegensatz zu diesem in ganz Europa Gültigkeit.

Mit der Anwendung Auriga kann ein Automatenmodell in KBF als Grafcet dargestellt werden.

4.3.1 Struktur eines Grafcet

Ein Grafcet beschreibt wesentlich zwei Aspekte einer Steuerung nach folgenden Regeln:

- die auszuführenden Aktionen (Befehle)
- den Ablauf der Ausführung

Deshalb ist ein Grafcet in zwei Teile gegliedert. Der erste Teil ist die Struktur, die den zeitlichen Ablauf des Prozesses zeigt. Dabei ist der Prozess in aufeinanderfolgende Schritte gegliedert. Der zweite Teil ist der Wirkungsteil, der die durchgeführten Aktionen bei jedem Schritt beschreibt (Siehe Abbildung 4.3).

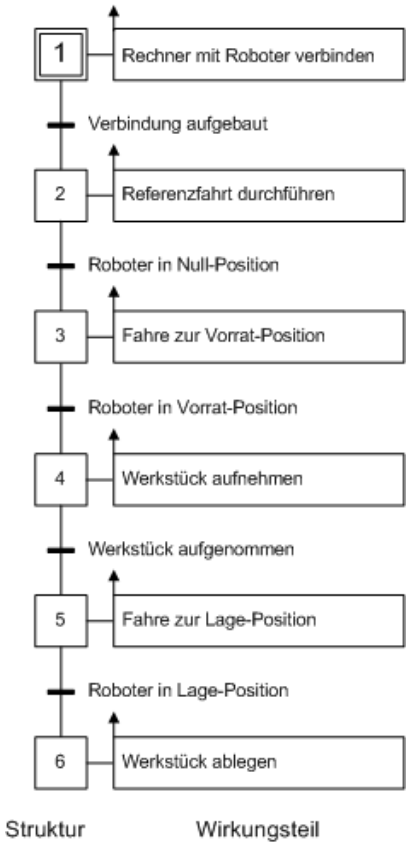


Abbildung 4.3 Grafcet für einen Transport eines Werkstücks [1]

4.3.2 Das Grundprinzip eines Grafcet

1) Abläufe werden unterteilt in
   - Schritte und
   - Transitionen,

   die sich abwechseln.
2) An die Schritte können beliebig viele Aktionen angeschlossen werden.
3) Abläufe können verzweigt und wieder zusammen geführt werden als
   - Alternativ-Verzweigung oder
   - Parallel-Verzweigung.

   Dabei ist Punkt 1 zu beachten!

4.3.3 Grafische Darstellung der Elemente

Schritte:

Die Abläufe werden in Schritte unterteilt. Jeder Schritt wird als Kästchen dargestellt, wobei das Quadrat dem Rechteck vorzuziehen ist. In der Mitte des Schrittfeldes muss eine alphanumerische Kennzeichnung stehen. Ein Schritt ist entweder aktiv – d.h. er wird gerade ausgeführt – oder inaktiv.

Abbildung 4.4 Beispiele für Schritte [1]

Startschritt:

Zu jeder Schrittkette gehört ein Startschritt. Der Startschritt kennzeichnet die Ausgangsstellung der Steuerung. In diesem Startschritt kennzeichnet die Steuerung unmittelbar nach dem Einschalten der Steuerung. Der Startschritt ist erkennbar am doppelten Rahmen.

Abbildung 4.5 Beispiel für einen Startschritt [1]

Transition und Transitionsbedingung:

Eine Transition ist die Verbindung von einem Schritt zum nächsten. Eine Transition wird auch als Übergang bezeichnet. Eine Transition wird durch eine Linie im rechten Winkel zur Verbindung zwischen den beiden Schritten dargestellt. Die Transition darf einen Transitionsnamen erhalten, der links von der Transition angeordnet wird.

Zu jeder Transition gehört eine Transitionsbedingung, auch Übergangsbedingung genannt. Die Transitionsbedingung ist ein logischer Ausdruck, der die Werte 1 (*true*) oder 0 (*false*) annehmen kann. Ist die Transitionsbedingung erfüllt, erfolgt die Transition auf den nachfolgenden Schritt. Die Transitionsbedingung steht auf der rechten Seite der Transition.

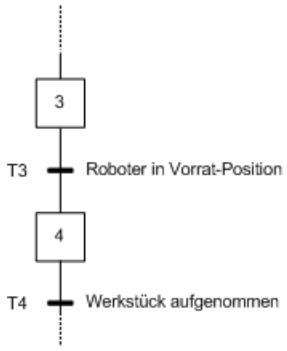


Abbildung 4.6 Beispiel einer Ablaufstruktur aus Schritten, Transitionen und Transitionsbedingungen [1]

Aktionen:

Jedem Schritt können eine oder mehrere Aktionen zugeordnet werden. Sie werden ausgeführt, wenn der Schritt aktiv ist. Eine Aktion wird als Rechteck mit beliebigem Seitenverhältnis dargestellt. Die Norm empfiehlt, für das Aktionsfeld und das Schrittsymbol die gleiche Höhe zu wählen. Aktionen können unterschiedliches Verhalten annehmen. Das Verhalten der Aktion wird durch entsprechende Zusätze dargestellt. Sind einem Schritt mehrere Aktionen zugeordnet, können sie auf unterschiedliche Weise grafisch dargestellt werden. Wichtiger Hinweis: Die Reihenfolge der Darstellung stellt keine zeitliche Reihenfolge dar!

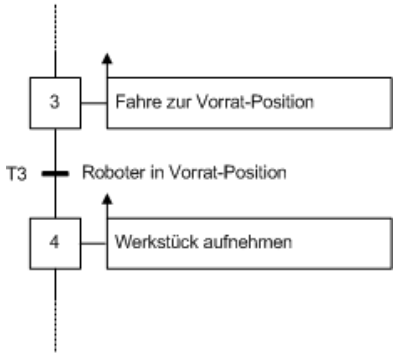


Abbildung 4.7 Beispieldarstellung eines Grafcet-Teils mit Aktionen [1]

Aktionen unterscheiden sich in der Art ihrer Ausführung. Es können zwei Aktionsarten unterschieden werden:

- kontinuierlich wirkende Aktionen:

  Diese werden über einen bestimmten Zeitraum ausgeführt. Endet der Zeitraum, wird die Aktion automatisch zurückgenommen.

- speichernde Aktionen:

  Diese werden zu einem bestimmten Zeitpunkt einmal ausgeführt. Dazu ist die präzise Angabe des Zeitpunkts unumgänglich. Zur Rücknahme des Befehls bedarf es eines weiteren Befehls.

4.3.4 Grafische Darstellung der Ablaufstrukturen

Durch Kombination der Elemente Schritt und Transition lassen sich drei Grundformen von Ablaufstrukturen erzeugen:

- Ablaufkette (linearer Ablauf)
- Ablaufverzweigung (alternative Verzweigung)
- Ablaufaufspaltung (parallele Verzweigung)

Unabhängig von der Form der Ablaufstruktur müssen sich Schritte und Transitionen immer abwechseln. Bearbeitet werden Ablaufstrukturen von oben nach unten.

Ablaufkette:

Eine Ablaufkette ist eine Folge von Schritten, in der:

- jeder Schritt nur eine nachfolgende Transition hat, mit Ausnahme des letzten,
- jeder Schritt nur eine vorangehende Transition hat, die durch einen einzigen Schritt der Ablaufkette freigegeben wird, mit Ausnahme des ersten.

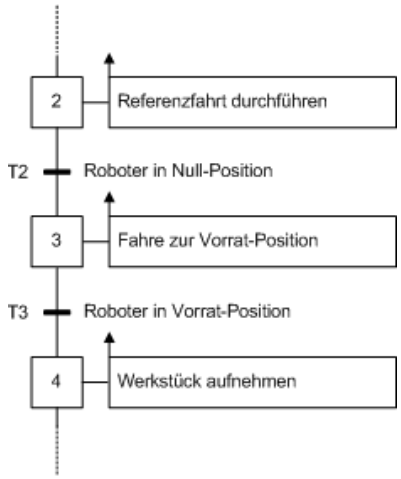


Abbildung 4.8 Beispiel eines linearen Ablaufs [1]

Alternative Verzweigung:

Bei der alternativen Verzweigung folgen zwei oder mehrere Transitionen auf einen Schritt. Es wird der Teilablauf aktiviert und bearbeitet, dessen Transitionsbedingung als erste erfüllt ist. Da bei der alternativen Verzweigung genau ein Teilablauf ausgewählt werden kann, müssen sich die Transitionsbedingungen gegenseitig ausschließen.

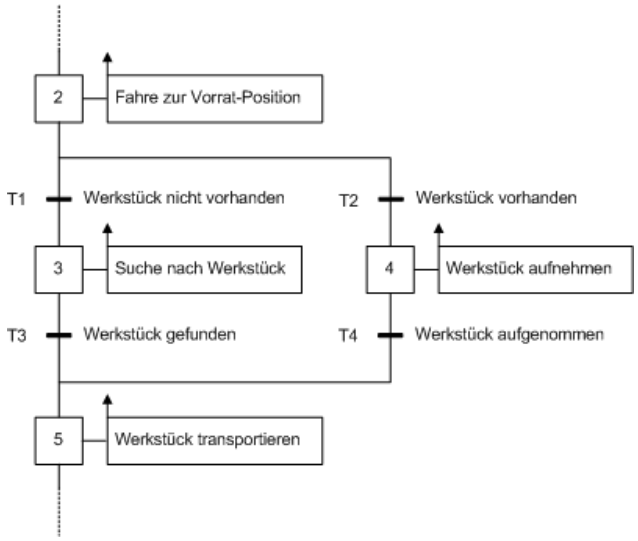


Abbildung 4.9 Beispiel einer alternativen Verzweigung [1]

Die Teilabläufe nach der Verzeigung können unterschiedlich lang sein. Ein Teilablauf kann bis auf eine einzige Transition reduziert werden (Überspringen von Schritten). Aus diesem Grund beginnt ein Alternativ-Zweig immer mit einer Transition und endet mit einer Transition. Die Nummerierung der Schritte ist dabei beliebig.

Parallele Verzweigung:

Bei der parallelen Verzweigung führt das Erfülltsein einer Transitionsbedingung zur gleichzeitigen Aktivierung von mehreren Teilabläufen. Die Teilabläufe werden gleichzeitig gestartet, aber unabhängig voneinander bearbeitet.

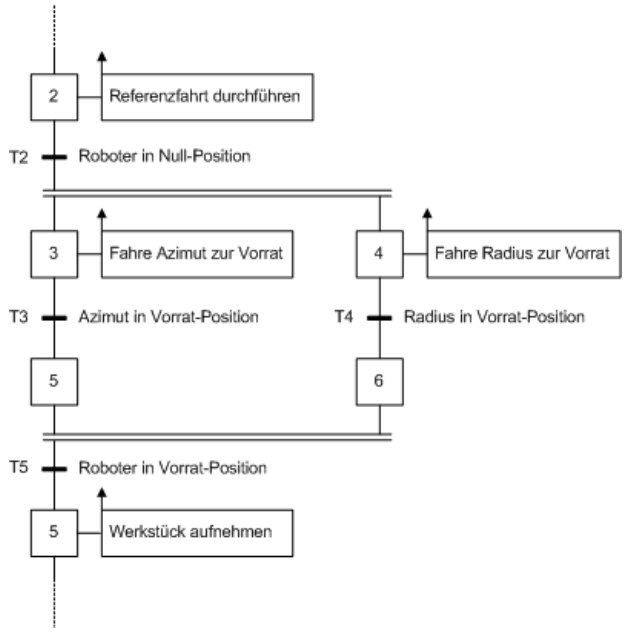


Abbildung 4.10 Beispiel einer parallelen Verzweigung [1]

Die Zusammenführung der Teilketten wird synchronisiert. Erst wenn alle parallelen Teilabläufe ganz bearbeitet sind, darf ein Übergang auf den Schritt unterhalb der Doppellinie – im Beispiel aus der Abbildung 4.10 auf Schritt 5 – erfolgen. Dazu muss unbedingt eine gemeinsame Transition erfüllt sein. Die Nummerierung der Schritte ist beliebig.

Rückführung:

Abläufe werden üblicherweise zyklisch durchlaufen, sie stellen also eine Schleife dar. Um die Schleifenstruktur darzustellen, muss eine Linie von unten nach oben verlaufen. Da diese Richtung der üblichen Richtung eines Ablaufs von oben nach unten entgegengesetzt ist, muss ein Pfeil angebracht werden.

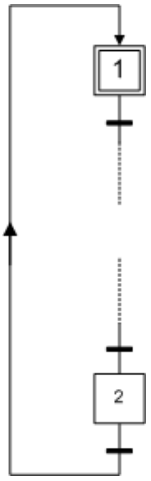


Abbildung 4.11 Beispiel zu einer Rückführung in einer Ablaufstruktur [1]

Sprünge:

Muss eine Wirkverbindung in einem Grafcet unterbrochen werden, weil der Grafcet kompliziert ist oder sich über mehrere Seiten erstreckt, muss an der Unterbrechungsstelle das Kennzeichen des Zielschrittes und die Nummer der Seite, auf der er erscheint, angegeben werden.

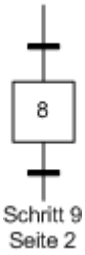


Abbildung 4.12 Beispiel für Unterbrechungsstelle in Ablaufstruktur [1]

5 3-Achs-Roboter von Fischertechnik

Für den Zweck der Entwicklungsarbeit wird ein 3-Achs-Roboter des Herstellers Fischertechnik verwendet. Die Vorteile des Roboters sind:

- Der Roboter hat die Grundstruktur eines Industrieroboters.
- Man kann die Funktionen für den Roboter mit verschiedenen Programmiersprachen programmieren.
- Es ist einfach, neue Komponenten einzubauen um die Funktionen des Roboters zu erweitern (siehe Kapitel 5.2.3).
- Der Roboter, der von der Firma Fischertechnik produziert wird, ist deutlich kostengünstiger als ein normaler Industrieroboter.

In den folgenden Kapiteln werden die Konstruktion sowie die Hardware- bzw. Softwarekomponenten des Roboters vorgestellt und beschrieben. Dabei werden ebenfalls die Grundfunktionen des Roboters erklärt.

5.1 Koordinatensystem

Der Roboter verfügt über drei bewegliche Achsen. Zwei davon sind lineare Achsen und die dritte ist eine Rotationsachse. Der Arbeitsbereich hat die Form eines Hohlzylinders. Deswegen wird für die Beschreibung ein zylindrisches Koordinatensystem verwendet. Die drei Achsen werden wie folgt definiert:

- *Die Radius-Achse* beschreibt den Abstand zwischen dem Greifer des Roboters und der Applikate-Achse (Radius ist nicht der Abstand vom Ursprung). Die Radius-Koordinate wird als *r* gezeichnet. Der Arbeitsbereich dieser Achse reicht von 0 bis 95 mm.
- *Die Azimut-Achse* beschreibt den Winkel zwischen der positiven Radius-Achse und der Projektion der Strecke AP auf die Boden-Ebene. Die Azimut-Koordinate

wird als *φ* gezeichnet. Der Arbeitsbereich dieser Achse reicht von 0 bis 200 Grad.

- *Die Applikate-Achse* beschreibt den orientierten Abstand zwischen dem Greifer des Roboters und der oberen Ebene, auf der der Null-Punkt liegt. Die Applikate-Koordinate wird als *z* gezeichnet. Der Arbeitsbereich dieser Achse reicht von minus 145 mm bis 0.

Die Zylinderkoordinate *P(r, φ, z)* gibt die Lage der Greifzange des Roboters an. Die Abbildung 5.1 stellt den Roboter und das Koordinatensystem dar.

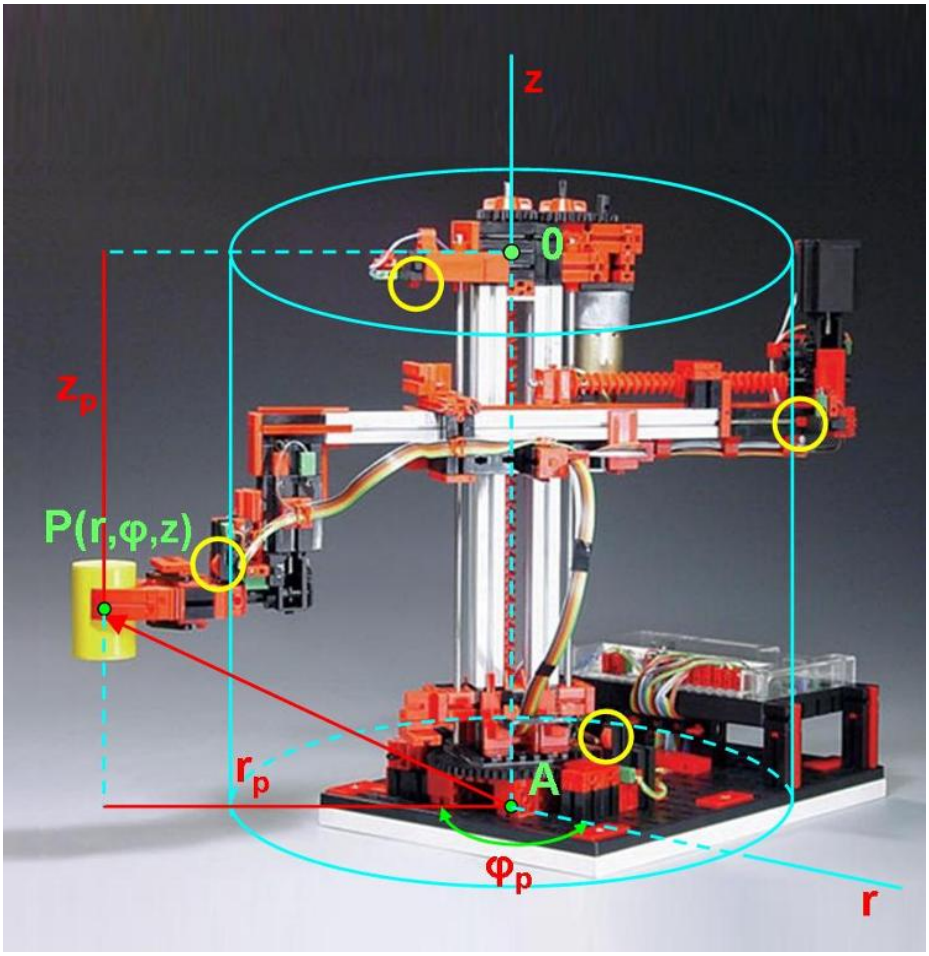


Abbildung 5.1 3-Achs-Roboter von Fischertechnik mit dem Koordinatensystem

Dieses vordefinierte Koordinatensystem vereinfacht die Erstellung eines Automatenmodells zur Robotersteuerung sowie die Programmierarbeit neuer Funktionen des Roboters, zum Beispiel die Bestimmung der aktuellen Position einer Achse bzw. der Fahrbefehl zu einer bestimmten Stelle. Die Null-Position des Roboters hat viel mit den Sensoren zu tun, die in Kapitel 5.2.1 vorgestellt werden.

5.2 Sensoren

5.2.1 Endlagenschalter

Ein Endlagenschalter hilft dabei, die Endposition einer Achse zu bestimmen. Der Roboter verfügt über 4 Endlagenschalter. Drei der Endlagenschalter befinden sich an einem Ende jeder Bewegungsachse. Sie dienen der Bestimmung der Null-Position der jeweiligen Achse. Der vierte Endlagenschalter wird am Greifer des Roboters angebaut und wird bei der maximalen Öffnung der Greifzange betätigt (siehe Abbildung 5.1, die Schalter sind durch Kreise markiert). Die Position, an der sich alle drei Achsen an ihren jeweiligen Endlagenschaltern befinden, wird als Null-Position definiert.

5.2.2 Schrittzähler

Für die Wegmessung jeder Achse wird jeweils ein Schrittzähler verwendet. Dabei handelt es sich um Schalter, die neben den Motoren montiert sind. Jeder Motor hat eine Getriebeachse, an der ein Impulsrad mit 4 Zähnen befestigt ist. Die Schrittzähler werden durch die Zähne des Impulsrads betätigt. Durch das Zählen der Betätigungen eines Schalters können die zurückgelegte Strecke entlang der Achsen bzw. die Öffnungsweite des Greifers bestimmt werden.

Da die gefahrene Strecke jeder Achse in Schritte gemessen wird, ist es erforderlich, die maximale Schrittzahl jeder Achse zu bestimmen. Nach der Messung sind folgende Werte bekannt:

- Radius-Koordinate: $0 \leq r \leq 161$ Schritte
- Azimut-Koordinate: $0 \leq \varphi \leq 260$ Schritte
- Applikate-Koordinate: -246 Schritte $\leq z \leq 0$

5.2.3 Ultraschallsensor

Der 3-Achs-Roboter verfügt anfangs über vier Endlagenschalter und vier Schrittzähler als Sensoren. Damit wird die Funktion des Roboters auf einfache Tätigkeiten beschränkt. Um den Roboter um neue Funktionen zu erweitern, z.B. ein Objekt zu erkennen oder zu suchen, wird ein Abstandsensor benötigt.

Der Hersteller Fischertechnik stellt den Ultraschallsensor (siehe Abbildung 5.2) als Abstandsensor zur Verfügung, der mit dem 3-Achs-Roboter kompatibel ist.

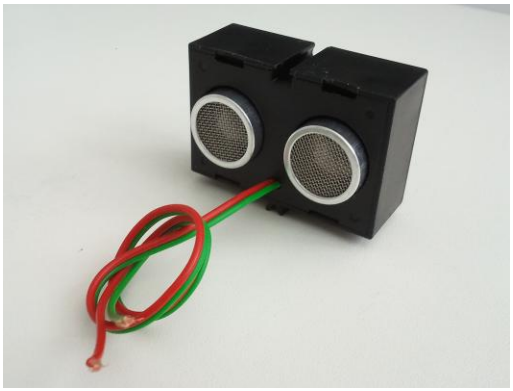


Abbildung 5.2 Ultraschallsensor von Fischertechnik

Ultraschall ist ein Schall mit Frequenzen, die oberhalb vom Hörbereich des Menschen liegen. Das umfasst Frequenzen zwischen 16 kHz (obere Hörschwelle) und 1,6 GHz. Der Sensor hat einen Sender und einen Empfänger, die wie zwei „Augen“ des Sensors aussehen. Der Sender sendet den Ultraschall aus. Falls ein Gegenstand im Erfassungsbereich des Sensors steht, wird der Ultraschall von dem Gegenstand reflektiert. Der Empfänger erfasst den reflektierten Schall. Durch die Messung der Zeit zwischen Sende- und Empfangsvorgang wird der Abstand zwischen dem Sensor und dem Gegenstand ermittelt.

Der Ultraschallsensor von Fischertechnik hat folgende technische Eigenschaften:

- Messbereich: von 2 bis 400 cm
- Einheit des zurückgegebenen Messabstandes: cm
- Erfassungsbereich:
  - Horizontale Erfassungswinkel: ca. 65°

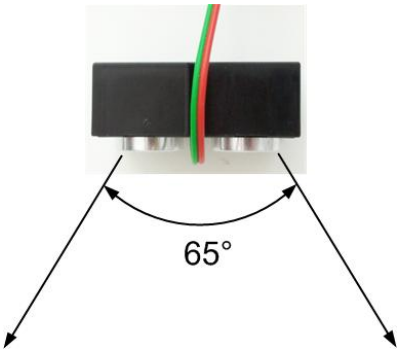


Abbildung 5.3 Horizontale Erfassungswinkel des Ultraschallsensors

- Vertikale Erfassungswinkel: ca. 43,5°

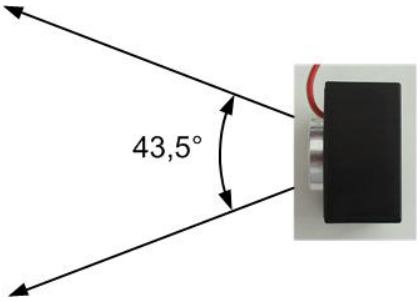


Abbildung 5.4 Vertikale Erfassungswinkel des Ultraschallsensors

- Der Anschluss erfolgt an Buchse D1 oder D2 der ROBO-Schnittstelle (Hier wird D1 verwendet). Die ROBO-Schnittstelle wird in Kapitel 5.4 vorgestellt.

Damit der Abstandsensor richtig funktioniert und mit größtmöglicher Genauigkeit arbeitet, muss der Sensor an eine geeignete Position am Roboter angebaut werden. Beim Anbau des Sensors sind folgende Punkte zu beachten:

- Der Sensor hat einen Erfassungsbereich. Alle Gegenstände, die darin liegen, werden vom Sensor erkannt. Deswegen muss der Sensor an einer Position angebaut werden, in der keine Komponenten des Roboters erfasst werden.
- Ein Zweck des neuen Sensors ist: Beobachtung vom Vorhandenzustand eines Objekts während des Transports. Deshalb muss der Abstandsensor nah am Greifer liegen, wo das Objekt festgehalten wird.
- Der Ultraschallsensor darf während der Arbeit des Roboters nicht mit anderen Teilen kollidieren.

- Der Abstand vom Ultraschallsensor zum einem festgehaltenen Objekt darf nicht kleiner als 2 cm sein. Andernfalls gibt der Sensor einen ungültigen Messwert zurück, da das Objekt außerhalb des Arbeitsbereichs des Sensors liegt.

Gemäß den oben genannten Bedingungen wird der Ultraschallsensor wie in der Abbildung 5.5 am Roboter montiert.

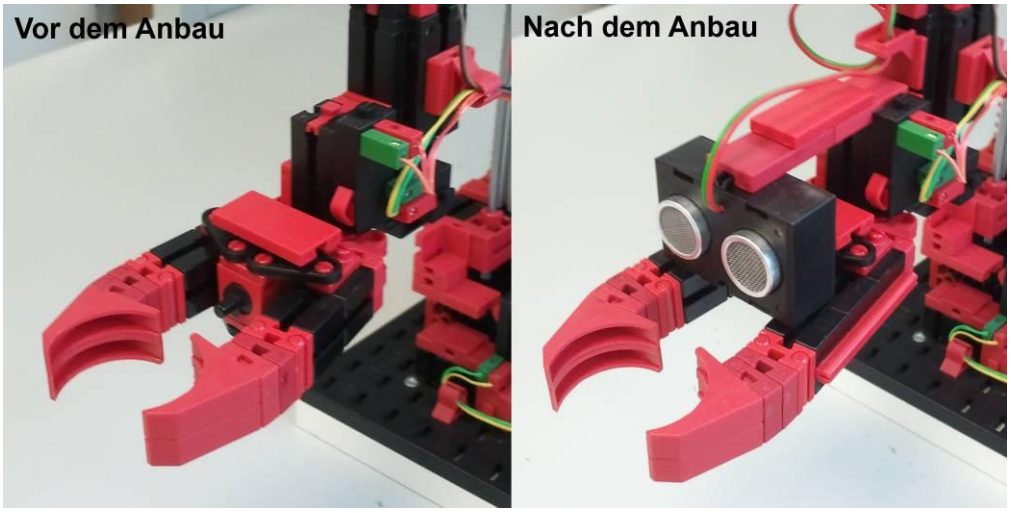


Abbildung 5.5 Anbau des Ultraschallsensors

5.3 Aktoren

Der Roboter hat insgesamt vier Gleichstrommotoren. Drei davon sind für die Bewegung in drei Achsen. Der vierte steuert die Greifzange. Nach der Angabe vom Hersteller des Roboters, verfügt jeder Motor über acht Geschwindigkeitsstufen (von 0 bis 7, dabei bedeutet die Stufe 0 „Motor stoppen“, die Stufe 7 liefert maximale Geschwindigkeit). Um zwischen der Schnelligkeit der Bearbeitung einer Aufgabe und der Sicherheit während der Bewegung des Roboter abzugleichen, wird die Stufe „Half“ bzw. Stufe 4 (Hälfte der maximalen Geschwindigkeit) als Standardgeschwindigkeit des Roboters benutzt. Nach mehreren Messungen wurden die Geschwindigkeiten der Stufe „Half“ jeder Achse wie folgend ermittelt:

- Motor auf r-Achse: 161 Schritte/11500 msec

  entspricht 95 mm/11500 msec

  entspricht 8,3 mm/sec

- Motor auf φ-Achse: 260 Schritte/8000 msec

entspricht 200 Grad/8000 msec

entspricht 25 Grad/sec

- Motor auf z-Achse: 246 Schritte/11800 msec

  entspricht 145 mm/11800 msec

  entspricht 12,3 mm/sec

Der vierte Motor steuert die Greifzange des Roboters. Der Greifer kann ein Werkstück mit einer Breite bis zu 50 mm festhalten.

Alle vier Motoren können gleichzeitig betrieben werden, ohne sich zu stören. Das ist sogar empfohlen, damit die Bearbeitung einer Aufgabe schneller geht.

5.4 ROBO-Schnittstelle

Die ROBO-Schnittstelle ermöglicht die Kommunikation zwischen einem Rechner und dem Roboter. Sie wandelt die Befehle der Software so um, dass beispielsweise Motoren angesteuert und Signale von Sensoren wie Impulstaste oder Endlagenschalter, verarbeitet werden können.

Die technischen Daten der ROBO-Schnittstelle sind:

- Stromversorgung 9V/1A
- Prozessor: 16 Bit, Taktfrequenz 16 MHz
- Speicher: 256 kByte RAM, 256 kByte Flash
- Analoge Ausgänge M1-M4: Anschlüsse für 4 Motoren mit 9V Gleichspannung, Dauerbetrieb 250 mA, kurzschlussfest
- Digitale Eingänge I1-I8: Anschlüsse für digitale Sensoren (Endlagenschalter, Schrittzähler). 9V Gleichspannung, Schaltschwelle für Ein/Aus bei ca. 2,6V, Eingangswiderstand ca. 10kΩ
- Weitere Ein- und Ausgänge:
  - Analoge Widerstandseingänge AX und AY
  - Analoge Spannungseingänge A1 und A2
  - Eingänge für Abstandssensoren D1 und D2

- Infrarot (IR)-Eingänge

- Schnittstellen USB/Seriell/IR: In unserem Fall wird die ROBO-Schnittstelle über eine USB-Schnittstelle mit dem Rechner verbunden.

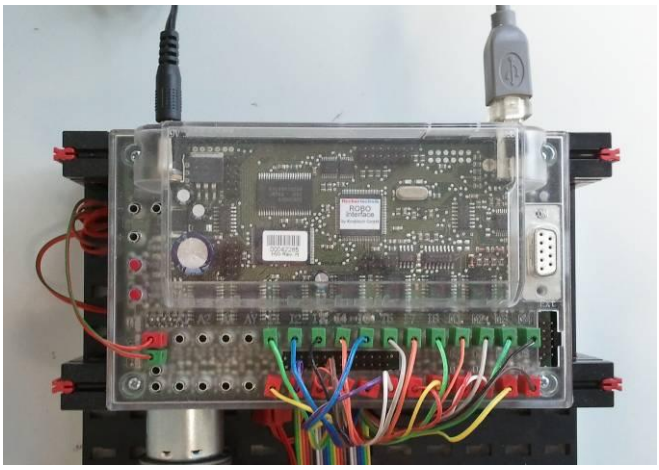


Abbildung 5.6 ROBO-Schnittstelle

5.5 Bibliotheksdateien von Fischertechnik

Der Hersteller des Roboters Fischertechnik stellt eine verwaltete und eine unverwaltete Bibliotheksdatei zur Verfügung, die einige Grundfunktionen für die Steuerung des Roboters bereitstellen. Die Dateien sind *FishFace2005.dll* (verwaltet) und *umFish40.dll* (unverwaltet).

5.5.1 Verwaltete Bibliotheksdatei *FishFace2005.dll*

Programmcode, der im *.NET* Framework unter der *.NET*-Laufzeitumgebung abläuft, wird als verwalteter Code bezeichnet. Mit der in C# geschriebenen Bibliotheksdatei *FishFace2005.dll* wird die Möglichkeit geboten, die Fischertechnik Schnittstelle mit einer *.NET*-2.0-Sprache zu programmieren.

5.5.2 Unverwaltete Bibliotheksdatei *umFish40.dll*

Im Gegensatz zu verwaltetem Code wird herkömmlicher Code als unverwalteter Code bezeichnet. Die Bibliothek *umFish40.dll* beinhaltet die Grundfunktionen für die Steuerung des 3-Achs-Roboters von Fischertechnik. Diese Funktionen kann Heron wegen

der unverwalteten Datei nicht direkt von *umFish40.dll* aufrufen. Die Steuerung tut das mittels eines Assembly, in dem die Datei *umFish40.dll* importiert wird. Die Tabelle 5.1 listet die Steuerungsbefehle auf, die in diesem Untersuchungssystem verwendet werden.

Befehl	Beschreibung
rbOpenInterfaceUSB	stellt eine Verbindung zu einer Schnittstelle an USB her
rbCloseInterface	beendet die Verbindung zur Schnittstelle
rbGetInput	liest den Zustand des angegebenen I-Einganges aus
rbGetDistanceValue	liest die aktuelle Werte in cm des angegebenen Ultraschallsensors aus
rbSetMotorEx	startet einen Motor einschließlich Angabe der Motorgeschwindigkeit
rbGetMotors	Einschaltstatus aller Motoren
rbRobMotor	startet einen Motor Der Befehl läuft asynchron und beendet sich bei Fahrschritte = 0 bzw. Endschalter selber.
rbGetCounter	liest die Schrittanzahl eines Schrittzählers

Tabelle 5.1 Liste der verwendeten Befehle

5.6 Anbindung des Roboters an Heron

In diesem Kapitel soll die Anbindung zwischen dem Roboter und der Steuerung Heron erläutert werden (siehe Abbildung 5.7). Angenommen wird entsprechend einer Aufgabe ein Automatenmodell in kanonischer Beschreibungsform erstellt (1). Das Automatenmodell liegt in Form einer XML-Datei vor. Diese Datei wird von Heron geladen (2). Heron ermittelt die verwendeten Sensoren-/Aktoren-Methoden, von denen manche zu externen Bibliotheksdateien gehören. Um diese Dateien aufzurufen und mit Heron zu verbinden, verwendet Heron eine Konfigurationsdatei (3), die ebenfalls in einem XML-Format vorliegt. Diese Datei hat die Funktion, die benötigten Bibliotheken zuzuordnen. Für die Anbindung des 3-Achs-Roboters lädt Heron die Datei *RobotModule.dll* (4). Diese Bibliothek wird aufgrund der Aufgabenstellung im Rahmen dieser Arbeit zusätzlich erstellt, denn die Aufgabe benötigt intelligente Sensoren/Aktoren, die in dieser Form in

den Bibliotheksdateien von Fischertechnik nicht existieren (Sie werden ausführlich im Kapitel 6 beschrieben). Die Bibliothek *umFish40.dll* (5) ist eine unverwaltete DLL-Datei. Die ist in der Datei *RobotModule.dll* importiert und wird automatisch von *.NET* nachgeladen. In *umFish40.dll* gibt es eine Initialisierungsfunktion, über die mithilfe des USB-Treibers *ftusb.sys* (6) die Verbindung zwischen der Steuerung und dem Roboter aufgebaut wird (7). Diese Funktion ist im Automatenmodell zuerst aufzurufen, bevor die Steuerungsbefehle ausgeführt werden können.

Außerdem kann die Anwendung Auriga (8) die XML-Datei auch lesen (9), in der das Automatenmodell liegt, und stellt den Automatenablauf als Grafcet dar. Durch eine Schnittstelle verbindet sich Auriga mit Heron (10), um den aktuellen Zustand des Automatenmodells auf der Softwareoberfläche anzuzeigen.

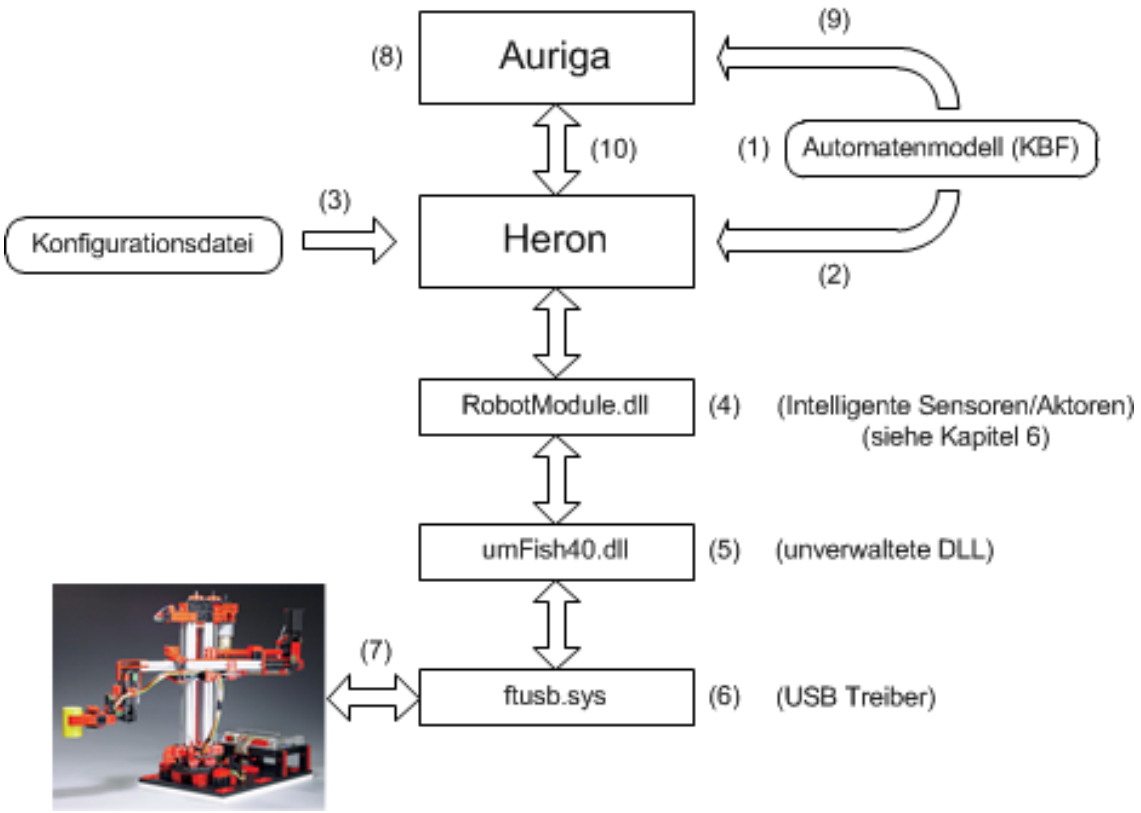


Abbildung 5.7 Anbindung des Roboters an Heron

6 Intelligente Sensoren/Aktoren

Intelligente Sensoren/Aktoren stellen eine stärker abstrahierte Sicht auf die Hardware des Roboters dar, die eine einfachere Verwendung innerhalb der Automatenmodelle ermöglicht. Zum Beispiel wird das Anfahren einer bestimmten Position in dem in Kapitel 5.1 vorgestellten Koordinatensystem angeboten.

6.1 Motivation

Wenn man ein Automatenmodell für die Robotersteuerung schreiben will, in dem es nur ein paar einfache Fahrbefehle gibt, reichen die Bibliotheksfunktionen von Fischertechnik. Für die Aufgabenstellung werden aber viel komplizierte Aktionen benötigt. Werden diese Aufgaben nur mit Grundfunktionen von Fischertechnik erledigt, wird das Automatenmodell sehr umfangreich. Damit erhöht sich auch die Wahrscheinlichkeit, beim Programmieren Fehler zu machen. Durch intelligente Sensoren/Aktoren reduziert sich die Anzahl der Befehle zum Positionieren. Das bringt eine bessere Übersichtlichkeit und vereinfacht den Vorgang beim Programmieren der Aufgabe.

6.2 Aufgabenstellung

Der Roboter hat nur vier Endlagenschalter, die die Endpositionen auf einer Seite jeder Achse festlegen. Es fehlen Sensoren für Bestimmung anderer Positionen, zum Beispiel der maximalen Position, die jeweils am anderen Ende einer Achse liegt. Auch die Zielposition, zu der sich eine Achse nach einem Fahrbefehl bewegen soll, ist wichtig. Die Speicherung der aktuellen Position des Roboters - auch während einer Bewegung - ist ebenfalls bedeutend.

Da ein Abstandssensor an den Roboter angebaut wurde, ist es erforderlich, eine Sensoren-Methode zu schreiben, die den Vorhandenzustand eines transportierten Objekts überwacht.

Nicht nur intelligente Sensoren sind notwendige, sondern auch intelligente Aktoren. Mit den vorhandenen Schrittzählern und Grundfunktionen in den Bibliotheksdateien von Fischertechnik sind Fahrfunktionen eingeschränkt. Die Achsen können damit nicht in mm oder in Grad, sondern nur in Schritten fahren. In der Praxis verwendet niemand Schrittzahlen, um Positionen anzugeben. Es werden Längen- bzw. Winkelmaße verwendet. Deshalb ist es umständlich, für jeden Fahrbefehl wieder den Abstand umzurechnen, dann den Roboter zu steuern. Nicht zu vergessen ist, dass der Roboter ein zylindrisches Koordinatensystem hat. Nur die Radius-Achse und Applikate-Achse fahren linear, die Azimut-Achse macht aber Umdrehungen. Deswegen sind die Umrechnungen unterschiedlich: Für die Azimut-Achse wird der Abstand von Grad in Schritte berechnet. Für die andere wird die Strecke in mm in Schritte umgewandelt.

Weitere wichtige Aktoren-Methoden sind: Referenzfahrt durchführen oder nach einem verlorenen Objekt suchen. Die beiden Funktionen benötigen viele Berechnungen und einen komplexen Algorithmus. Es ist vorteilhaft, dafür einzelne Methoden anzubieten, die in einem Automatenmodell mehrfach verwendet werden können.

In der Steuerungstechnik spielen Sicherheitsanforderungen eine große Rolle. Man soll bestmöglich an alle Störungsfälle, Ausnahmen usw. denken und eine Lösung für jeden Fall finden. Viele Lösungen kann man mit intelligenten Sensoren/Aktoren realisieren. Im Folgenden sind einige beispielhafte Sicherheitsanforderungen für den Roboter aufgeführt:

- Ungültige Werte sollen erkannt werden, z.B. Zielpositionen die nicht im Arbeitsbereich der jeweiligen Achse liegen.
- Verbindungsverlust und Verlust eines Objekts während des Aufnehmens oder Transportierens ist zu erkennen.
- Die Motoren sollen beim Erreichen der Fahrtgrenzen automatisch gestoppt werden.

Um die intelligente Sensoren/Aktoren zu schreiben, wird die objektorientierte Programmiersprache C Sharp (C#) von Microsoft verwendet. Das Programm läuft auf der Software-Plattform .NET. Die Quellcodes des Projekts zur Erstellung der intelligenten Sensoren/Aktoren sind auf der beigefügten DVD abgelegt.

6.3 Sensoren-Methoden

In Tabelle 6.1 werden alle intelligenten Sensoren-Methoden, die entsprechend der Aufgabenstellung erstellt werden, aufgelistet und beschrieben.

Sensoren-Methode	Beschreibung
ConnectStatus()	hat keinen Parameter Diese Sensoren-Methode gibt den Verbindungszustand zwischen der Steuerung und der ROBO-Schnittstelle an Heron zurück. Achtung: Falls die Verbindung während Arbeit des Roboters abgebrochen wird, stoppt der Roboter automatisch unabhängig von dieser Methode. Die Rückmeldung ist trotzdem notwendig, damit die Steuerung diese Ausnahme erkennt.
InZeroPosition(axis)	hat einen Parameter *axis* Diese Sensoren-Methode gibt zurück, ob sich eine Achse in Null-Position befindet (an ihrem Endlagenschalter). Der Parameter *axis* bestimmt, für welche Achse die Methode verwendet wird.
InMaxPosition(axis)	hat einen Parameter *axis* Diese Sensoren-Methode gibt zurück, ob sich eine Achse in der Endlage gegenüber der Null-Position befindet. Der Parameter *axis* bestimmt, für welche Achse die Methode verwendet wird.
GetPosition(axis)	hat einen Parameter *axis* Diese Sensoren-Methode liest aktuelle Position (in Schritten) einer Achse aus und rechnet sie in mm bzw. Grad um. Der Parameter *axis* bestimmt, für welche Achse die Methode verwendet wird.
ReachedTargetPosition (axis)	hat einen Parameter *axis*. Diese Sensoren-Methode gibt zurück, ob eine Achse ihre Zielposition erreicht. Der Parameter *axis* bestimmt, für welche Achse die Methode verwendet wird.
ObjectExist()	hat keinen Parameter. Diese Sensoren-Methode gibt zurück, ob sich gerade ein Objekt in der Greifzange befindet. Diese Methode funktioniert basierend auf

	die Messung des Abstands vom Ultraschallsensor zum Objekt.
ObjectFound()	hat keinen Parameter. Diese Sensoren-Methode gibt zurück, ob sich das verlorene Objekt im Suchfeld befindet (siehe die Beschreibung der Aktoren-Methode LookForObject)

Tabelle 6.1 Intelligente Sensoren

6.4 Aktoren-Methoden

In Tabelle 6.2 werden die intelligenten Aktoren beschrieben.

Aktoren-Methode	Beschreibung
Connect()	hat keinen Parameter Diese Aktoren-Methode stellt die Verbindung zum ROBO-Schnittstelle her.
Disconnect()	hat keinen Parameter Diese Aktoren-Methode trennt die Verbindung zum ROBO-Schnittstelle.
Move(axis, dir)	hat zwei Parameter *axis* und *dir* Der Parameter *axis* bestimmt, für welche Achse die Methode verwendet wird. Der Parameter *dir* bestimmt die Fahrtrichtung. Diese Methode steuert eine Achse oder den Greifer in negative/positive Richtung zu fahren.
StartReferenceDrive(axis)	hat einen Parameter *axis*, der bestimmt, für welche Achse die Methode verwendet wird Diese Aktoren-Methode startet eine Referenzfahrt einer Achse. Referenzfahrt bedeutet, dass eine Achse in Richtung zu ihrem Endlagenschalter fährt. Die Achse stoppt automatisch, wenn ihr Endlagenschalter aktiviert ist. Falls alle drei bewegliche Achsen sich in Endlage befinden, steht der Roboter in der Null-Position.
MoveTo(axis, target)	hat zwei Parameter *axis* und *target* Diese Aktoren-Methode startet eine Fahrt einer Achse. Die

	Achse stoppt automatisch, wenn ihre Zielposition erreicht wird. Der Parameter *axis* bestimmt, für welche Achse die Methode verwendet wird. Der Parameter *target* ist die Zielposition in mm/Grad. (Algorithmus der Methode wird in Kapitel 6.4.1 erläutert)
StopMotors()	hat keinen Parameter. Diese Aktoren-Methode stoppt alle Motoren des Roboters.
LookForObject(axis)	hat einen Parameter *axis* Mit dieser Methode kann der Roboter nach einem Objekt in entweder Azimut- oder Radius-Achse suchen und zu der Koordinate des Objekts fahren. Da der Roboter nur in einem bestimmten Bereich arbeitet, muss auch ein Suchbereich definiert werden. Der Roboter kann nur das Objekt erkennen und aufnehmen, das sich in dem Suchbereich befindet. Der Parameter *axis* bestimmt, für welche Achse die Methode verwendet wird. (Algorithmus der Methode wird in Kapitel 6.4.2 erläutert).

Tabelle 6.2 Intelligente Aktoren

6.4.1 Algorithmus der Aktoren-Methode *MoveTo(axis, target)*

Ziel der Erstellung dieser Methode ist Vereinfachung mehrerer Funktionsaufrufen in einen konkreten Funktionsbefehl. Nachdem die Methode aufgerufen wird, werden folgende Schritte ausgeführt:

- Entsprechend der Achse, die von dem Parameter *axis* angegeben ist, wird die Zielposition von mm bzw. Grad in Schritte umgerechnet. Der Umrechnungsfaktor ist eine von den Eigenschaften des Roboters (siehe Kapitel 6.5).
- Durch den Vergleich zwischen der aktuelle Position und der Zielposition des Greifers wird die Fahrrichtung bestimmt (positive oder negative Fahrrichtung).
- Das Steuersignal wird zum richtigen Motor gesendet.
- Während des Ablaufs wird gleichzeitig die aktuelle Position des Greifers gespeichert und überwacht. Wenn die aktuelle Position mit der Zielposition übereinstimmt, wird den Motor gehalten.

- Die Aktoren-Methode *ReachedTargetPosition(axis)* entsprechender Achse wird auf *true* gesetzt.

6.4.2 Algorithmus der Aktoren-Methode *LookForObject(axis)*

Der Fischertechnik Roboter kann nur mit Gegenstände umgehen, die sich in seinem Arbeitsbereich befinden. Nach dieser Beschränkung wird angenommen, dass ein verlorenes Werkstück während eines Transports auf ein vordefiniertes Suchfeld fällt, dessen Fläche im Arbeitsbereich des Roboters und auf der Bodenebene liegt. Deswegen sind die Azimut- und Radius-Koordinate des verlorenen Werkstücks vor dem Suchvorgang unbestimmt, die Applikate-Koordinate ist aber konstant. Falls sich das Werkstück außerhalb des Suchbereichs befindet, wird der Roboter es nicht erkennen. Außerdem kann der Roboter das verlorene Werkstück von anderen Gegenständen nicht unterscheiden. Deshalb dürfen keine anderen Objekte innerhalb des Suchfeldes liegen.

Aufgrund der gleichbleibenden Höhe des Suchfeldes wird der Suchablauf in zwei Abschnitte geteilt: Das Suchen auf der Azimut-Achse, gefolgt von dem Suchen auf der Radius-Achse. Eine Eingabe der Achse Applikate in den Parameter axis der Methode *LookForObject(axis)* gilt als ungültig. Anders als die Aktoren-Methode *MoveTo(axis)*, hat diese Aktoren-Methode für einzelne Achsen verschiedene Algorithmen. Sie werden in die folgenden Kapitel beschrieben.

6.4.2.1 Der Suchvorgang in der Azimut-Achse

- Der Roboter fährt zur Anfangsstelle des Suchvorgangs, die zum Beispiel nah an der Endlage der Azimut-Achse und ein wenig höher als die Bodenebene liegt (siehe Abbildung 6.1).

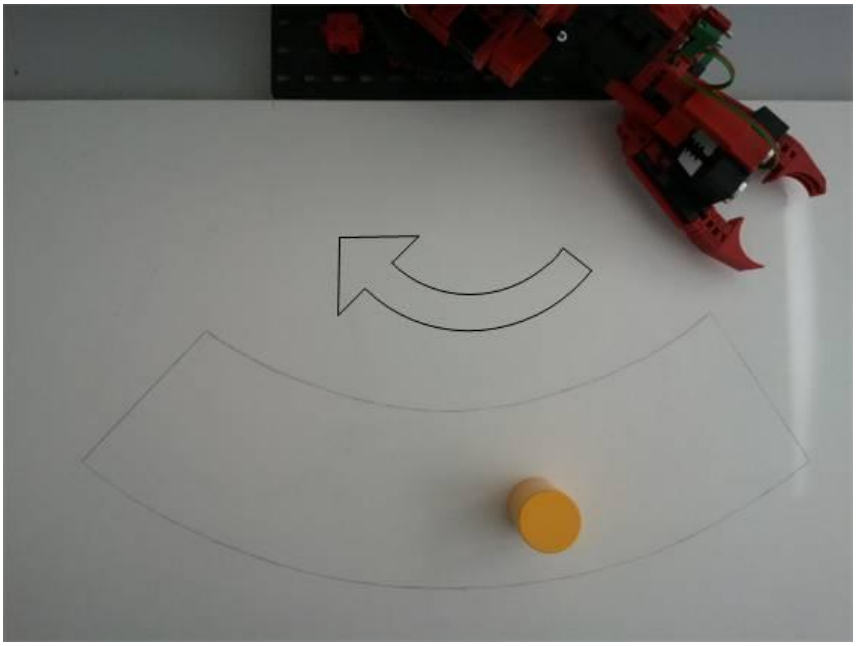


Abbildung 6.1 Anfangsstelle des Suchvorgangs in der Azimut-Achse

- Dann bewegt sich der Greifer langsam in die positive Richtung (siehe Abbildung 6.1). Gleichzeitig misst er den Abstand von sich selbst zum nächsten Gegenstand, der vor dem Sensor steht. Die Methode wurde programmiert, damit der Roboter nur ein Objekt erkennt, das im Suchfeld liegt (durch den Abstandsmesswert).
- Der Ultraschallsensor hat einen Erfassungswinkel. Deswegen identifiziert der Sensor das Objekt schon ab dem Zeitpunkt, zu dem das Objekt den Erfassungswinkel betritt (siehe Abbildung 6.2). Der Roboter erkennt das Objekt, denn der Messwert des Abstands zum Objekt sagt, dass es einen Gegenstand im Suchfeld gibt. Da das Suchfeld nur für das gesuchte Objekt reserviert wird, muss es das richtige Objekt sein.

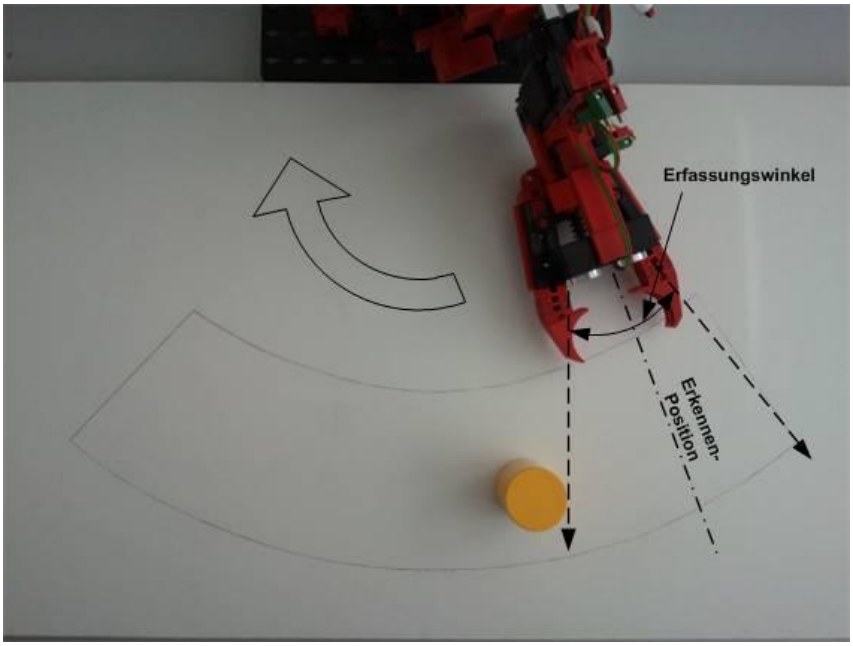


Abbildung 6.2 Position, wo der Abstandsensor das Objekt erkennt

- An dieser Stelle wird die aktuelle Azimut-Koordinate als Erkennen-Position gespeichert. Der Roboter fährt weiter in die positive Richtung der Azimut-Achse.
- An dem Zeitpunkt, wenn das Objekt nicht mehr in dem Erfassungswinkel liegt, wird die aktuelle Azimut-Koordinate als Verlassen-Position gespeichert (siehe Abbildung 6.3). Jetzt wird der Roboter gehalten.

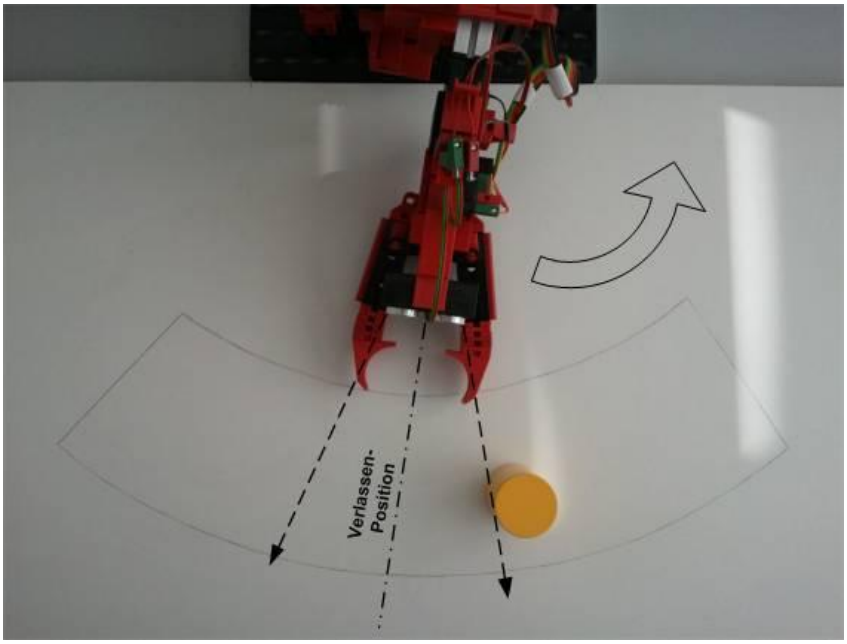


Abbildung 6.3 Position, wo der Abstandsensor das Objekt verlässt

- Das untere Diagramm in der Abbildung 6.4 stellt die Messwerte des Ultraschall-sensors im Suchvorgang in der Azimut-Achse dar.

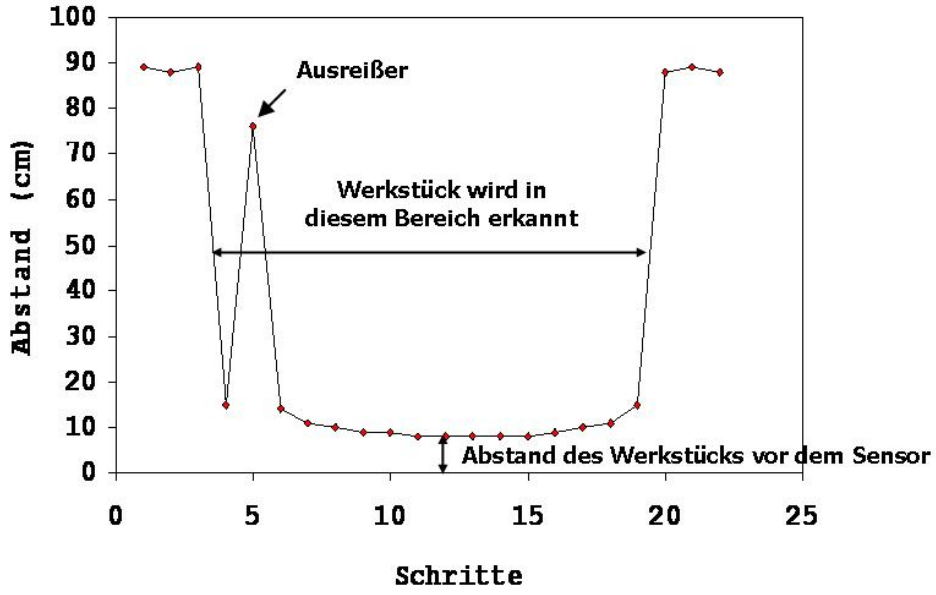


Abbildung 6.4 Messwert-Diagramm für Suchvorgang in der Azimut-Achse

- Im Diagramm kann man die Ungenauigkeit des Sensors sehen. Im Bereich, wo das Werkstück erkannt wird, sollten die Messwerte keinen großen Unterschied

haben. Das Auftreten von Ausreißern ist aber nicht zu vermeiden. Deswegen wird die richtige Azimut-Koordinate des Objekts wie folgt berechnet.

- Die genaue Azimut-Koordinate des Objekts ist der Mittelwert zwischen Erkennen- und Verlassen-Position. Der Roboter fährt zurück in die negative Richtung bis er die Koordinate des Objekts erreicht. Der Sensor ObjectFound wird auf 1 gesetzt. Dann endet der Suchvorgang in der Azimut-Achse.
- Falls es kein Objekt auf dem Suchfeld gibt, nachdem der Roboter zum Ende des Felds fährt, wird der Sensor ObjectFound auf 0 gesetzt. Die Steuerung kann darauf reagieren.

6.4.2.2 Der Suchvorgang in der Radius-Achse

Der Vorgang wird erst nach einem erfolgreichen Suchen in Azimut-Achse durchgeführt. Mit anderen Worten: Das Objekt steht jetzt vor dem Greifer. Das Suchen beinhaltet folgende Schritte:

- Der Greifer fährt langsam in die positive Richtung der Radius-Achse (zum Objekt). Gleichzeitig misst der Ultraschallsensor den Abstand von sich selbst zum Werkstück (siehe Abbildung 6.5).

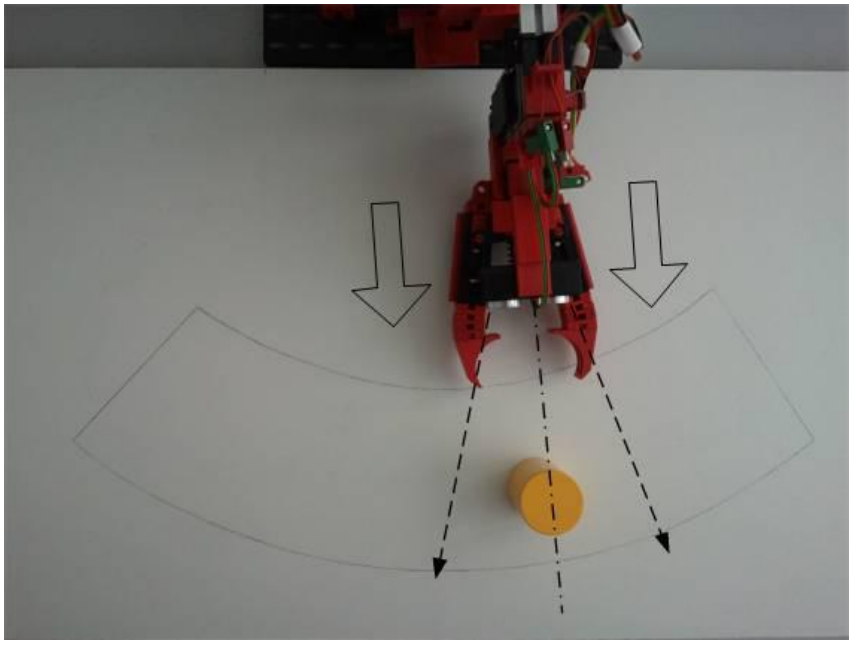


Abbildung 6.5 Anfangsstelle des Suchvorgang in der Radius-Achse

- Wenn der Greifer sich nah an dem Objekt befindet (Abstand ist klein genug), stoppt der Roboter. Der Sensor *ReachedTargetPosition* signalisiert dies der Steuerung. Der Suchvorgang ist beendet (siehe Abbildung 6.6).

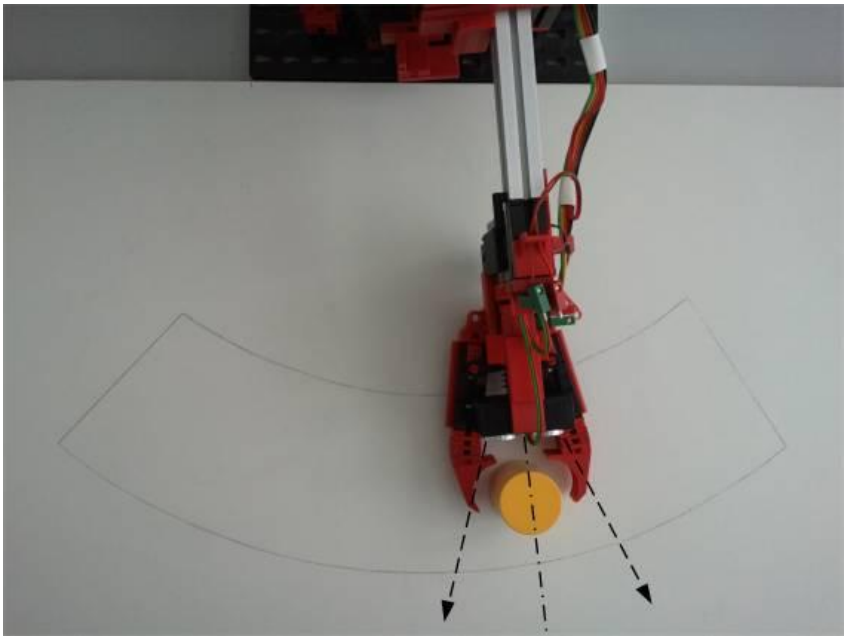


Abbildung 6.6 Anfangsstelle des Suchvorgang in der Radius-Achse

- Das folgende Diagramm (Abbildung 6.7) zeigt die Messwerte der Abstände in dem Suchablauf in der Radius-Achse.

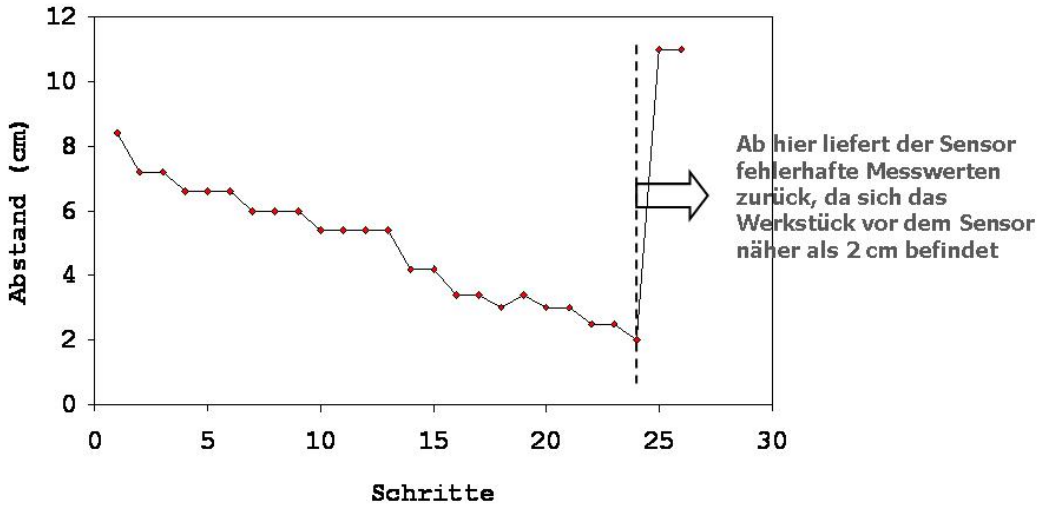


Abbildung 6.7 Messwert-Diagramm für Suchvorgang in der Radius-Achse

6.5 Konfigurierbare Eigenschaften des Roboters

Zu den Eigenschaften des Roboters gehören zum Beispiel Motorgeschwindigkeit, Umrechnungsfaktor für Positionen, maximale Position usw. Nach Bedarf sollen diese Werte geändert werden können. Wenn sie als Parameter in Quellcodes der Bibliotheksdatei deklariert werden, ist die Veränderung kompliziert. Die Lösung sind konfigurierbare Eigenschaften, die in einer externen Datei definiert sind. Die Bibliotheksfunktionen können diese Datei lesen, um die Eigenschaften zu ermitteln. Andererseits hat ein Benutzer die Möglichkeit, die charakteristischen Eigenschaften des Roboters zu lesen und ändern, ohne die Struktur der Quellcodes verstehen zu müssen oder zu gefährden. Der andere Vorteil der konfigurierbaren Eigenschaften ist, dass die Module für andere Roboter verwendet werden können, die eine gleichartige Struktur, aber andere Geometrie haben.

In diesem Fall wird eine XML-Datei verwendet, um die Eigenschaften zu beschreiben. Jede Achse besitzt folgende Eigenschaften (siehe Tabelle 6.3):

Eigenschaft	Beschreibung
MaximumRange	maximale Position in mm/Grad, bis zu der eine Achse fahren darf
MaximumSteps	maximale Position in Schritten, bis zu der eine Achse fahren darf

Factor	Umrechnungsfaktor für Berechnung zwischen mm/Grad und Schritte jeder Achse Dieser Wert ist eine Dezimalzahl.
DriveID	Identifizierungsnummer eines Motors (1 bis 4)
Negative	negative Fahrrichtung einer Achse; wird als Nummer angegeben (1 oder 2)
Positive	positive Fahrrichtung einer Achse; wird als Nummer angegeben (1 oder 2)
Speed	Fahrgeschwindigkeit einer Achse Es gibt insgesamt 8 Stufen (von 0 bis 7, 7 ist am schnellsten).
TerminalSwitch	Identifizierungsnummer eines Endlagenschalters (1 bis 4)

Tabelle 6.3 Eigenschaften jeder Achse des Roboters

7 Experimentelle Untersuchung

Nach Abbildung 3.1 ist das Untersuchungssystem fast komplett. Es fehlt nur noch das Automatenmodell, damit das ganze System funktioniert. Dafür muss eine Testaufgabe verfasst werden. Da es sich um eine Robotersteuerung handelt, soll die Aufgabe anspruchsvoll sein und dem Steuerungsprozess entsprechen. Mit der Testaufgabe sollen alle Grundfunktionen und neuen Funktionen des Roboters geprüft werden. Außerdem ist die Kommunikation zwischen dem Roboter und der Steuerung Heron während der Durchführung der Aufgabe zu beobachten und analysieren. Alle typischen Kenngrößen der Steuerungstechnik sind zu erfassen.

7.1 Beschreibung der Testaufgabe

Diese Aufgabe simuliert eine Transportarbeit beim Entladen eines LKWs. In dem Arbeitsraum des Roboters befinden sich ein LKW, ein Lager und zwischen den beiden liegt das Suchfeld, dessen Fläche vordefiniert wurde. Die Aufgabe ist, Werkstücke vom LKW zum Lager zu bringen. Allerdings ist der große und wichtige Teil der Aufgabe, dass der Roboter Ausnahmen und Fehler, die während der Arbeit des Roboters auftreten könnten, erkennen und behandeln kann.

Der Roboter funktioniert in 2 Betriebsarten: Handbetrieb und Automatikbetrieb. Die Wahl der Betriebsarten und den Wechsel zwischen diesen kann der Benutzer selbst bestimmen. Im Hand-betrieb kann der Benutzer alle Motoren des Roboters freiwillig steuern. Im Automatikbetrieb führt der Roboter die Tätigkeiten selbsttätig aus, die in dem Automatenmodell geschrieben wurden.

Der Benutzer steuert den Roboter durch die Schalter, die als einfache Bedienoberflächen (siehe Abbildung 7.1) und durch ein Modul von Heron erzeugt werden. Die Schalter sind zustandsbehaftet, d.h. einmaliges Betätigen überführt den Schalter in den Zustand „gedrückt“. Ein nochmaliges Betätigen gibt den Schalter wieder frei.

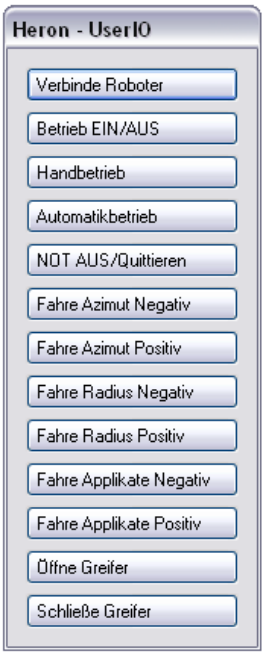


Abbildung 7.1 Steuerschalter als Bedienoberflächen

Die Tabelle 7.1 stellt alle Schalter und ihre Funktionen dar.

Schalter	Beschreibung
Verbinde Roboter	baut die Verbindung zwischen Heron und ROBO-Schnittstelle auf Nur nach die Aktivierung der Kommunikation sind die anderen Schalter verwendbar.
Betrieb EIN/AUS	schaltet den Hauptbetrieb ein und aus Die Betriebsarten können nur im Ein-Zustand ausgewählt werden. Auf Sicherheitgründe hält der Roboter im Automatikbetrieb nach der Deaktivierung des Schalters (Betrieb ausschalten) nicht sofort, sondern erst nachdem die nächste Referenzfahrt beendet wird.
Handbetrieb	aktiviert den Handbetrieb oder wechselt die Betriebsart Automatikbetrieb in Handbetrieb
Automatikbetrieb	aktiviert den Automatikbetrieb oder wechselt die Betriebsart Handbetrieb in Automatikbetrieb
NOT AUS/Quittieren	Der Schalter funktioniert nur im Automatikbetrieb. Der Schalter gilt im Zustand „gedruckt“ als NOT AUS und im Zustand „nicht gedruckt“ als Quittieren. Bei NOT AUS wird der Automatikbetrieb

	gestoppt und alle Motoren sofort angehalten. Bei „Quittieren“ wird der Behandlungszustand von NOT AUS verlassen.
Fahre Azimut Negativ	fährt die Achse Azimut in die negative Richtung bei der Betätigung des Schalters und stoppt diese Achse bei der Freigabe Der Schalter funktioniert nur im Handbetrieb.
Fahre Azimut Positiv	fährt die Achse Azimut in die positive Richtung bei der Betätigung des Schalters und stoppt diese Achse bei der Freigabe Der Schalter funktioniert nur im Handbetrieb.
Fahre Radius Negativ	fährt die Achse Radius in die negative Richtung bei der Betätigung des Schalters und stoppt diese Achse bei der Freigabe Der Schalter funktioniert nur im Handbetrieb.
Fahre Radius Positiv	fährt die Achse Radius in die positive Richtung bei der Betätigung des Schalters und stoppt diese Achse bei der Freigabe Der Schalter funktioniert nur im Handbetrieb.
Fahre Applikate Negativ	fährt die Achse Applikate in die negative Richtung bei der Betätigung des Schalters und stoppt diese Achse bei der Freigabe Der Schalter funktioniert nur im Handbetrieb.
Fahre Applikate Positiv	fährt die Achse Applikate in die positive Richtung bei der Betätigung des Schalters und stoppt diese Achse bei der Freigabe Der Schalter funktioniert nur im Handbetrieb.
Öffne Greifer	öffnet den Greifer bei der Betätigung des Schalters und stoppt diesen bei der Freigabe Der Schalter funktioniert nur im Handbetrieb.
Schließe Greifer	schließt den Greifer bei der Betätigung des Schalters und stoppt diesen bei der Freigabe Der Schalter funktioniert nur im Handbetrieb.

Tabelle 7.1 Liste der Schalter für Robotersteuerung

Am Anfang der Aufgabe wird durch die Steuerung des Benutzers die Verbindung aufgebaut. Dann kann der Hauptbetrieb gestartet werden. Danach befindet sich der Betrieb in dem Bereitzustand. Der Benutzer kann zwischen dem Handbetrieb oder dem

Automatikbetrieb entscheiden. Während des Laufens eines Betriebs kann die Betriebsart gewechselt werden. Der Wechsel von Handbetrieb in Automatikbetrieb erfolgt nur, wenn alle Motoren nicht laufen. Die Umschaltung in umgekehrte Richtung passiert erst, wenn sich der Roboter in der Null-Position befindet.

Im Handbetrieb können alle Achsen und der Greifer parallel laufen. In einer Achse kann die Fahrtrichtung durch die Betätigung des Schalters, der diese Achse in umgekehrte Richtung mit der aktuellen Richtung fährt, einfach wechseln lassen. Beim Ausschalten von dem Handbetrieb und von dem Hauptbetrieb werden alle Motoren sofort angehalten.

Im Automatikbetrieb hat der Benutzer im Gegensatz vom Hand-betrieb wenige Einflussmöglichkeiten, da der Roboter alles automatisch macht, was im Modell steht. Der Automatikbetrieb beinhaltet folgende Schritte:

1) Durchführung der Referenzfahrt, um den Roboter in die Null-Position zu bringen; Wartezeit von 3 Sekunden nach der Referenzfahrt

2) Fahrt zum LKW

    a) Falls sich kein Werkstück im LKW befindet: Zurück zu Schritt 1)

    b) Falls sich ein Werkstück im LKW befindet: Aufnahme des Werkstücks

3) Transport des Werkstücks zum Lager

- Falls der Roboter das Werkstück während des Transports verliert: Suchen auf der Bodenebene nach dem Werkstück

    a. Falls sich das Werkstück im Suchfeld befindet: Aufnahme des Werkstücks und Fortsetzen des Schritts 4)

    b. Falls sich das Werkstück nicht im Suchfeld befindet: Zurück zum Schritt 1)

4) Ablegung des Werkstücks am Lager

5) Rückkehr zu Schritt 1)

Der Wechsel und das Beenden des Automatikbetriebs werden erst nach einer erfolgreichen Referenzfahrt ausgeführt. Das bedeutet wenn der Schalter „Handbetrieb“ betätigt wird oder der Schalter „Automatikbetrieb“ freigegeben wird, macht der Roboter seine Arbeit weiter bis er den Null-Punkt erreicht, danach wird die Betriebsart gewechselt bzw. wird der Automatikbetrieb beendet. Während des ganzen Transportvorgangs wird

der Verbindungs-zustand ständig überwacht. Falls die Verbindung abgebrochen wird (z.B. das USB-Kabel wird ausgezogen), stoppt der Roboter sofort. Damit der Roboter weiter arbeiten kann, muss die Verbindung zwischen Heron und ROBO-Schnittstelle zuerst wieder aufgebaut werden (das Kabel wird wieder eingesteckt und der Schalter „Verbinde Roboter“ wird betätigt).

7.2 Anfangsbedingungen

Die Aufgabe setzt voraus, dass folgende Anfangsbedingungen erfüllt werden:

- Die LKW- und Lagerposition müssen festgelegt werden und dürfen während des Aufgabenablaufs nicht geändert werden.
- Die Grenze des Suchfeldes muss auf die Bodenebene gezeichnet werden.
- Auf dem Suchfeld darf kein Gegenstand liegen, sonst kann der Ultraschallsensor das zu transportieren Werkstück mit anderen Gegenständen nicht unterscheiden.
- Das Suchfeld muss möglichst eben sein, sonst werden die Messwerte des Ultraschallsensors ungenau.
- Die Person, die den Roboter steuert, darf nicht zu nah vor dem Abstandsensor stehen oder den Raum oberhalb des Suchfelds betreten.

7.3 Steuerungsentwurf

Für die Erstellung eines Automatenmodells sind Petrinetze sehr hilfreich. Mithilfe von Petrinetzen ist es einfacher, den Ablauf des Modells nachzuvollziehen. Darüber hinaus vereinfacht es die Abbildung des Automatenmodells auf KBF. In dem Petrinetz sind die Beziehungen zwischen Zustände des Automatenablaufs deutlicher zu erkennen. Außerdem sieht man hier, wo eine bestimmte Ausnahme auftreten kann. Es ist noch ein Vorteil, dass ein Petrinetz ähnliche Aufbaustrukturen wie GRAFCET hat.

Das ganze Automatenmodell wird in drei Vorgänge geteilt: den Betriebskopf, den Handbetrieb und den Automatikbetrieb. Der Vorgangsablauf wird in einem Petrinetz gezeichnet.

In einem Petrinetz sieht man meistens, dass es neben den Zustände Aktionen und neben den Transitionen Ereignisse gibt. Tabelle 7.2 und Tabelle 7.3 listen die Aktionen und Ereignisse, die in den Petrinetzen der Aufgabe verwendet werden, und ihre Beschreibungen, damit die Abläufe der Petrinetze veranschaulicht werden.

Aktion	Beschreibung
Verbinde_Roboter	verbindet Heron mit der ROBO-Schnittstelle
Trenne_Roboter	trennt Verbindung zwischen Heron und ROBO-Schnittstelle
Starte_Zeitgeber_1	startet den Zeitgeber 1, der 3 Sekunden lang läuft
Starte_Zeitgeber_2	startet den Zeitgeber 2, der eine Sekunde lang läuft
Fahre_<Achse>_Negativ	fährt eine Achse in die negative Richtung
Fahre_<Achse>_Positiv	fährt eine Achse in die positive Richtung
Fahre_<Achse>_Referenz	fährt eine Achse zur Null-Position
Fahre_<Achse>_LKW	fährt eine Achse zur Position des LKWs
Fahre_<Achse>_Lager	fährt eine Achse zur Position des Lagers
Fahre_<Achse>_Transport	fährt eine Achse zur Position zum Transport eines Objekts
Fahre_<Achse>_ Suchposition	fährt eine Achse zur Position vom Anfang des Suchvorgangs
Suche_Azimut	sucht nach dem verlorenen Objekt in der Achse Azimut
Suche_Radius	sucht nach dem verlorenen Objekt in der Achse Radius
Stoppe_<Achse>	stoppt eine Achse
Stoppe_Motoren	stoppt alle Achsen und den Greifer
Freigebe_<Schalter>	gibt einen Schalter frei
Freigebe_Fahrenschalter	gibt alle Schalter frei, die im Handbetrieb zum Steuern der Motoren verwendet werden

Tabelle 7.2 Liste der Aktionen

Ereignis	Beschreibung
Verbindung_EIN/ Verbindung_AUS	Verbindung zwischen Heron und ROBO-Schnittstelle ist ein- bzw. ausgeschaltet
Betrieb_EIN/ Betrieb_AUS	Der Hauptbetrieb ist ein- bzw. ausgeschaltet.
Handbetrieb_EIN/ Handbetrieb_AUS	Der Handbetrieb ist ein- bzw. ausgeschaltet.
Automatikbetrieb_EIN/ Automatikbetrieb_AUS	Der Automatikbetrieb ist ein- bzw. ausgeschaltet.
NOT_AUS	Der Schalter NOT AUS ist betätigt.
Quittiert	Der Schalter NOT AUS ist freigegeben.
<Achse>_Negativ_aktiviert	Der Schalter, der eine Achse in die negative Richtung fährt, ist betätigt.
<Achse>_Negativ_deaktiviert	Der Schalter, der eine Achse in die negative Richtung fährt, ist freigegeben.
<Achse>_Positiv_aktiviert	Der Schalter, der eine Achse in die positive Richtung fährt, ist betätigt.
<Achse>_Positiv_deaktiviert	Der Schalter, der eine Achse in die positive Richtung fährt, ist freigegeben.
Wartezeit_1_vorbei	Die Laufzeit des Zeitgebers 1 ist abgelaufen.
Wartezeit_2_vorbei	Die Laufzeit des Zeitgeber 2 ist abgelaufen.
<Achse>_in_Endlage	Eine Achse hat ihre Endlage erreicht.
<Achse>_in_Ziel	Eine Achse hat ihre Fahrziel erreicht.
Objekt_vorhanden	Ein Objekt befindet sich im LKW oder vor dem Greifer.
Objekt_verloren	Kein Objekt befindet sich im LKW oder vor dem Greifer.
Objekt_gefunden	Das verlorene Objekt liegt innerhalb des Suchfeldes.

Tabelle 7.3 Liste der Ereignisse

- Das Feld <Achse> in Tabelle 7.2 und Tabelle 7.3 kann mit folgenden Achsennamen ersetzt werden:
  - Azimut
  - Radius
  - Applikate
  - Greifer
- Das Feld <Schalter> in der Tabelle 7.2 kann mit den Schalternamen in der Tabelle 7.1 ersetzt werden.

Folgende sind die Petrinetzen des Automatenmodells:

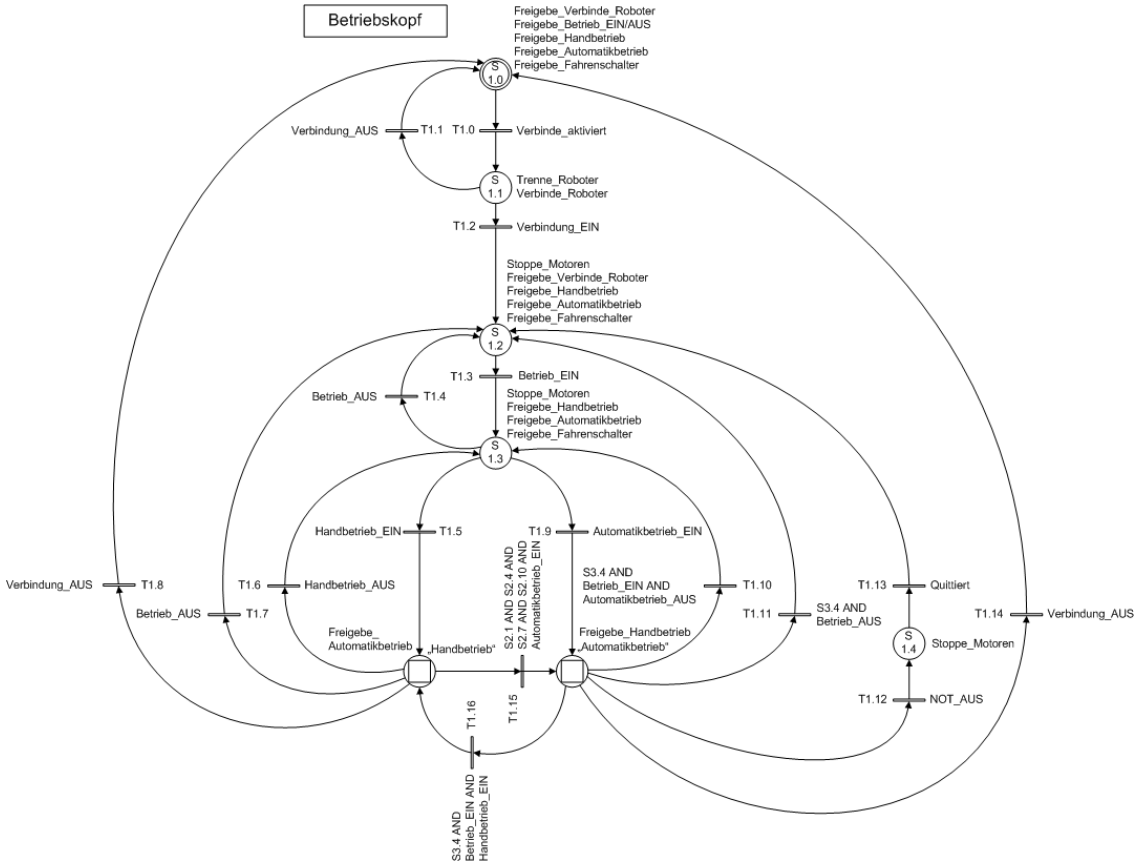


Abbildung 7.2 Modell des Betriebskopfs

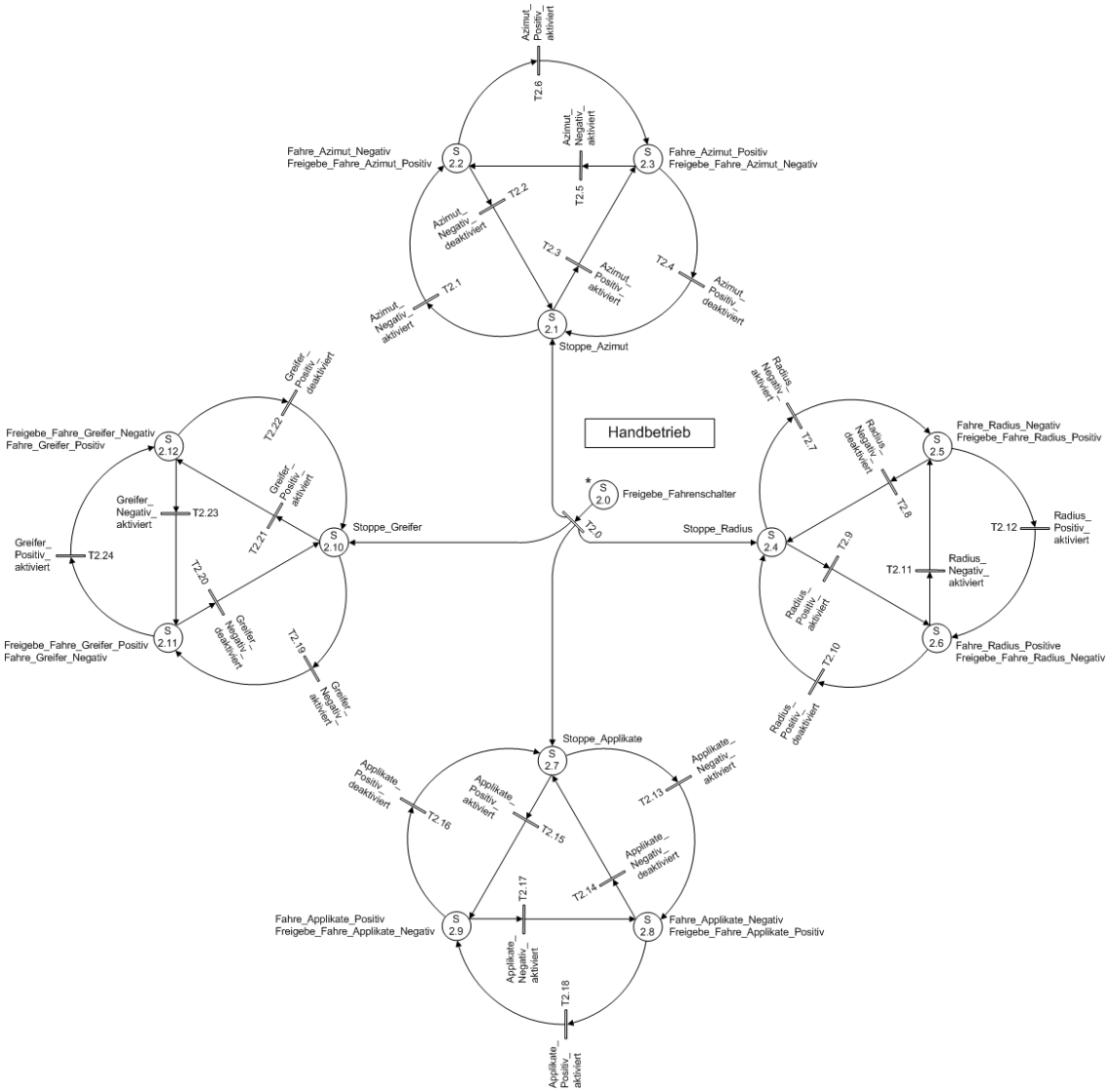


Abbildung 7.3 Modell des Handbetriebs

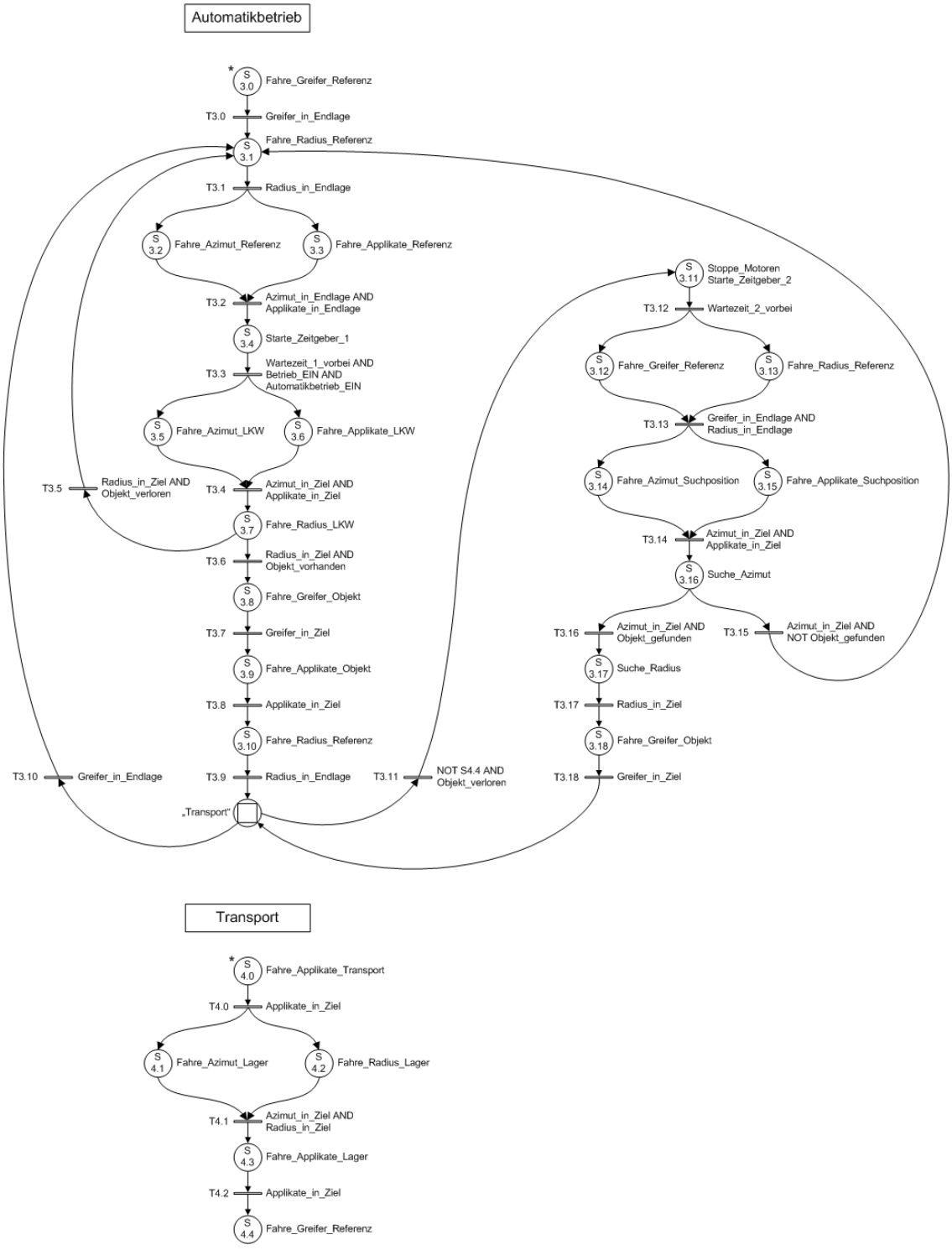


Abbildung 7.4 Modell des Automatikbetriebs

7.4 Abbildung auf kanonische Beschreibungsform (KBF)

Nachdem die Petrinetze der gesteuerten Strecke entworfen wurden, werden diese auf KBF abgebildet. Die Abbildung 7.5 ist ein Ausschnitt aus dem Petrinetz des Modells und wird gezeigt, um den Abbildungsvorgang deutlicher zu erklären.

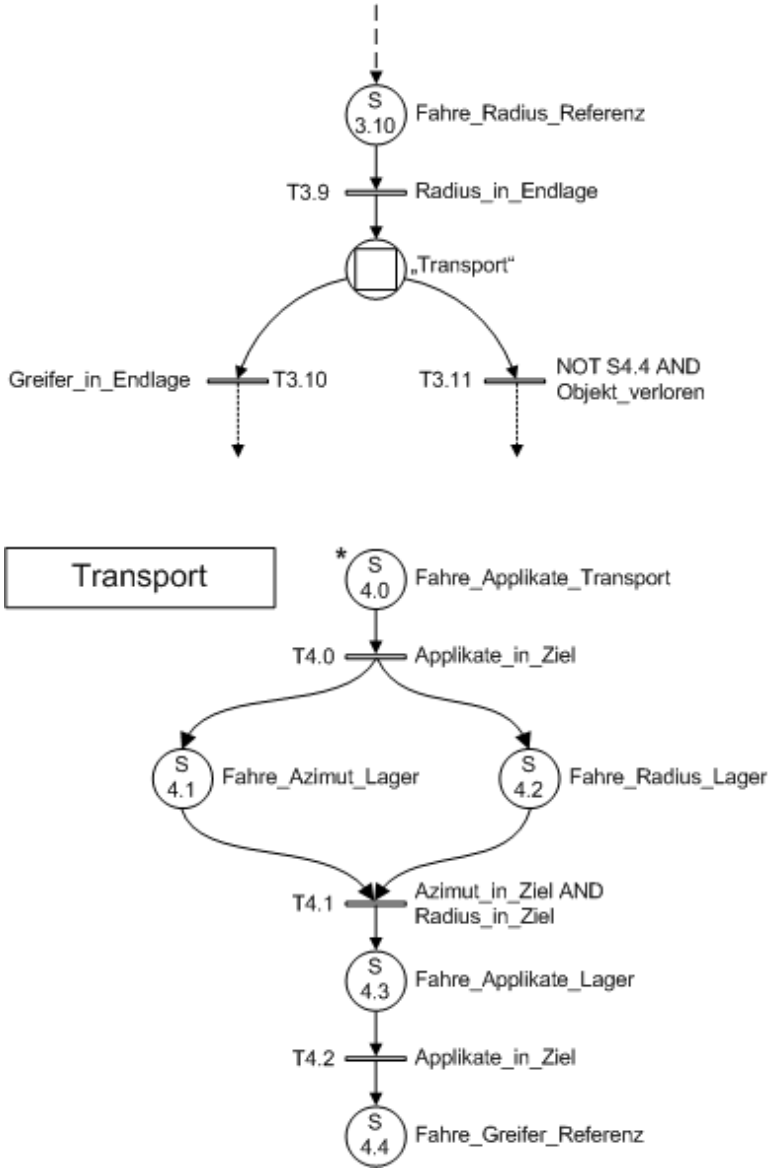


Abbildung 7.5 Ausschnitt aus Petrinetz des Modells

Der Zustand *S3.10* soll jetzt auf KBF abgebildet werden. Codeauszug 7.1 stellt die Abbildung dar.

```
<State id="S3.10">
  <Actions>
```

```
      <Action ref="Fahre_Radius_Referenz" />
   </Actions>
   <Enables>
      <Transition ref="T3.9" />
   </Enables>
</State>
```

Codeauszug 7.1 Abbildung eines Zustandes auf KBF

Ein neuer Zustand fängt mit seiner Identifikation in der ersten Zeile an. In der dritten Zeile von Codeauszug 7.1 steht der Name der Aktion (*Fahre_Radius_Referenz*), die bei der Aktivierung des Zustandes durchzuführen ist. In der 6. Zeile wird die Transitionsnamen (*T3.9*) eingegeben, zu dieser Transition der Zustand *S3.10* hinführt (siehe Abbildung 7.5). Eine Aktion kann das Ergebnis von einem oder mehreren Aktoren sein. Die Beziehung zwischen Aktion und Aktor wird in Codeauszug 7.2 vorgestellt.

```
<Action id="Fahre_Radius_Referenz">
   <Actor ref="actor_Fahre_Radius_Referenz" />
</Action>
```

Codeauszug 7.2 Abbildung einer Aktion auf KBF

Codeauszug 7.3 zeigt, aus welchem Modul der Aktor abgerufen wird und welche Parameter er hat.

```
<Actor id="actor_Fahre_Radius_Referenz" module="Robot"
operation="StartReferenceDrive">
   <Parameter name="axis" value="Radius" />
</Actor>
```

Codeauszug 7.3 Abbildung eines Aktors auf KBF

In der ersten Zeile wird die Identifikation des Aktors eingegeben, dann das Modul und die verwendete Aktoren-Methode. Falls diese Aktoren-Methode Parameter besitzt, muss der Wert jedes Parameters eingefüllt werden.

Im Petrinetz in der Abbildung 7.5 sieht man, dass das Ereignis *Radius_in_Endlage* neben der Transition *T3.9* liegt. Die Transition *T3.9* schaltet nur wenn dieses Ereignis vorkommt und der Zustand *S3.10* vor der Transition *T3.9* gesetzt ist. Die Abbildung der Transition auf KBF wird im Codeauszug 7.4 durchgeführt.

```
<Transition id="T3.9" condition="C3.9">
   <Actions></Actions>
   <NextStates>
      <State ref="Transport" />
   </NextStates>
</Transition>
```

Codeauszug 7.4 Abbildung einer Transition auf KBF

In der ersten Zeile des Codeauszug 7.4 sind die Identifikation der Transition (*T3.9*) und der Bedingungsname (*C3.9*), mit dieser Bedingung die Transition *T3.9* verbunden ist. In der 4. Zeile ist der nächste Zustand Transport, zu dem die Transition *T3.9* hinführt (siehe Abbildung 7.5).

Bis jetzt ist das Ereignis Radius_in_Endlage der Transition *T3.9* noch nicht aufgetaucht. Codeauszug 7.5 zeigt, dass dieses Ereignis in der Bedingung *C3.9* liegt.

```
<Condition id="C3.9">
  Radius_in_Endlage
</Condition>
```

Codeauszug 7.5 Abbildung einer Bedingung auf KBF

Wie ein Sensor ein Ereignis erzeugt, beschreibt Codeauszug 7.6.

```
<Event id="Radius_in_Endlage">
  <Sensor ref="Endlagenschalter_Radius" type="true" />
</Event>
```

Codeauszug 7.6 Abbildung eines Ereignisses auf KBF

Das Ereignis *Radius_in_Endlage* wird erzeugt, wenn der Sensor mit dem Namen *Endlagenschalter_Radius* einen Wert *true* zurückgibt.

Wie ein Aktor wird ein Sensor auch von einem Modul abgerufen. Codeauszug 7.7 zeigt diesen Vorgang in KBF.

```
<Sensor id="Endlagenschalter_Radius" module="Robot" operation="InZeroPosition">
  <Parameter name="axis" value="Radius" />
</Sensor>
```

Codeauszug 7.7 Abbildung eines Sensors auf KBF

Die erste Zeile beschreibt, dass der Sensor *Endlagenschalter_Radius* aus dem Modul Robot abgerufen und die Sensoren-Methode *InZeroPosition* ausführt. Diese Sensoren-Methode hat einen Parameter *axis* und sein Wert ist *Radius*.

In Abbildung 7.5 gibt es den übergeordneten Zustand Transport, der mehrere normale Zustände beinhaltet (Zustände *S4.0* bis *S4.4*). Codeauszug 7.8 formuliert die Abbildung dieses übergeordneten Zustandes auf KBF.

```
<SuperState id="Transport">
  <Actions></Actions>
  <Enables>
```

```
      <Transition ref="T3.10"/>
      <Transition ref="T3.11"/>
    </Enables>
    <ContainedStates>
      <State ref="S4.0"/>
      <State ref="S4.1"/>
      <State ref="S4.2"/>
      <State ref="S4.3"/>
      <State ref="S4.4"/>
    </ContainedStates>
    <Activates>
      <State ref="S4.0"/>
    </Activates>
  </SuperState>
```

Codeauszug 7.8 Abbildung eines übergeordneten Zustands

Der Name des übergeordneten Zustandes wird in der ersten Zeile initialisiert. Dieser Zustand führt zu zwei Transitionen *T3.10* und *T3.11* zu und besitzt die Zustände von *S4.0* bis *S4.4* (Zeile 8 bis 12). Falls dieser Zustand besetzt ist, wird der Zustand anschließend auch aktiviert.

Das Beispiel hat die Abbildung aller Komponenten von Petrinetz auf KBF geklärt. Bei anderen Stellen in den Petrinetzen des Modells werden auch gleiche Regeln verwendet. Die vollständige KBF ist auf der beigefügten DVD abgelegt.

7.5 Auswertung der Ergebnisse

Nachdem die Testaufgabe durchgeführt wurde (siehe das Video auf der beigefügten DVD oder das Link auf YouTube [13]), sind die Ergebnisse auszuwerten.

7.5.1 Gesamtüberblick des Experiments

Jeden Aufgabenschritt hat der Roboter richtig, wie gefordert, durchgeführt. Während Arbeit des Roboters werden die Ereignisse rechtzeitig gemeldet. Sowohl einzelne Aktoren als auch mehrere Aktoren gleichzeitig funktionieren ohne Probleme. Nach jeder Ausnahme reagiert der Roboter sehr schnell und behandelt diese rechtzeitig und sinnvoll. Unterschiede zwischen den Zuständen bei Arbeit des Roboters in der Realität und in der GRAFCET-Darstellung lassen sich kaum feststellen.

7.5.2 Positionierungen

Da die sich Motoren des Roboters nach Schritten bewegen, muss die eingegebene Zielposition jeder Achse in Schritte umgerechnet werden. Durch die Umrechnung fallen die Ziffern nach dem Komma weg, denn die Schritte müssen ganze Zahlen (Integer) sein. Daraus ergibt sich unter Umständen eine kleine Abweichung zwischen Soll- und Ist-Position (ca. 1 mm).

Beim Behandeln der Ausnahme: „Objekt verloren“ entsteht ebenfalls eine kleine Abweichung von der Positionsbestimmung. Die Ausnahme könnte an jeder Position passieren, die zurückgegebene Position ist aber eine genaue Schrittanzahl. Die Abweichung in diesem Fall beträgt ca. 1 mm.

7.5.3 Reaktionszeiten

Es gibt verschiedene Reaktionszeiten:

- bei der Meldung eines Ereignisses
- bei der Durchführung einer Aktion
- bei der Behandlung einer Ausnahme

Die Reaktionszeiten in allen oben genannten Fällen bewegen sich im Millisekunden-Bereich. Es gibt so gut wie keine Zeitverzögerung.

8 Zusammenfassung und Ausblick

Kann eine Steuerung für verschiedene Industrieroboter verwendbar sein? Ein Teil dieser Arbeit ist, die Antwort zu finden. Es hält aber nicht nur beim Test die Zusammenarbeit von der Steuerung und dem Roboter, sondern auch die Fähigkeit, neue Funktionen für den Roboter zu definieren, soll geprüft werden.

Für diese experimentelle Untersuchung werden der Roboter und die dafür verwendete Steuerung von zwei anderen Firmen eingesetzt. Die Steuerung Heron ist von der Firma IFS Informationstechnik entwickelt. Der 3-Achs-Roboter ist ein Produkt des Unternehmens Fischertechnik. Die neue Funktionen des Roboters wurden mit der Programmiersprache C# in Visual Studio 2005 erstellt. Folgende Punkte wurden durch die Arbeit erledigt:

- Untersuchung des mechanischen Aufbaus der Sensoren/Aktoren und der Grundfunktionen des Roboters vom Hersteller Fischertechnik
- Entwicklung intelligenter Sensoren/Aktoren, um neue Funktionen für den Roboter zu realisieren
- Definition einer Transportaufgabe, die vom Roboter gelöst werden soll: Durch diese Aufgabe sollen alle Funktionen des Roboters sowie die Kommunikation zwischen dem Roboter und der Steuerung getestet werden.
- Entwurf der Steuerung als zeitbehaftetes, interpretiertes Petrinetz und Darstellung in der kanonischen Beschreibungsform (KBF).
- Test der Steuerung am Roboter; Erfassen typischer Kenngrößen der Steuerungstechnik (z.B. Reaktionszeiten).

Weiterzuentwickeln kann die Lösung des Problems breiter als nur im Robotik-Bereich anwendbar sein, z.B. in Anlagensteuerung oder Produktion. Obwohl der Test am Roboter in der Hauptsache gut gelaufen ist, hat dieser Roboter nur eine einfache Struktur und Funktionalitäten. Um einen normalen Industrieroboter oder eine Anlage zu steuern, benötigt Heron wahrscheinlich mehr Module mit erweiterten Funktionen und Senso-

ren/Aktoren, zum Beispiel binäre Funktionen (ODER, Exklusive-ODER), Zeitfunktionen (Taktgeber), Zähler, Löschtransition (bestimmte Schritte werden verlassen wenn diese Transition schaltet), Anweisungstransition, Triggertransition usw. Außerdem, falls es um einen großen Fertigungsprozess mit vielen Robotern geht, sollte die Steuerung die Möglichkeit haben, gleichzeitig alle Roboter zu steuern und die Abläufe dieser Roboter zu synchronisieren.

Literaturverzeichnis

[1] Festo Didactic GmbH & Co.KG: *GRAFCET*, Bildungsverlag Eins GmbH (2008), ISBN: 9783427548676.

[2] Gevatter, Hans-Jürgen; Grünhaupt, Ulrich: *Handbuch der Mess- und Automatisierungstechnik im Automobil Fahrzeugelektronik, Fahrzeugmechatronik*, Springer (2005), ISBN: 9783540212058.

[3] Hesse, Stefan; Schnell, Gerhard: *Sensoren für die Prozess- und Fabrikautomation*, Vieweg + Teubner Verlag (2008), ISBN: 9783834804716.

[4] Litz, Lothar: *Grundlagen der Automatisierungstechnik*, Oldenbourg Wissenschaftsverlag (2005), ISBN: 9783486273830.

[5] Lunze, Jan: *Automatisierungstechnik*, Oldenbourg Wissenschaftsverlag (2003), ISBN: 9783486274301.

[6] Lunze, Jan: *Ereignisdiskrete Systeme: Modellierung und Analyse dynamischer Systeme mit Automaten, Petrinetzen und Markovketten*, Oldenbourg Wissenschaftsverlag (2006), ISBN: 9783486580716.

[7] Wörn, Heinz; Brinkschulte, Uwe: *Echtzeitsysteme: Grundlagen, Funktionsweisen, Anwendungen*, Springer (2005); ISBN: 9783540205883.

[8] Meiners, Ulfert: *Steuerungstechnik*, Skriptum zur Vorlesung WS 2008/2009, Hochschule für Angewandte Wissenschaften Hamburg.

[9] Pritschow, G.: *Steuerungstechnik der Werkzeugmaschinen und Industrieroboter*, Skriptum zur Vorlesung SS 1997, Universität Stuttgart.

[10] *Ereignisgesteuerte Architektur*, http://de.wikipedia.org/wiki/Ereignisgesteuerte_Architektur.

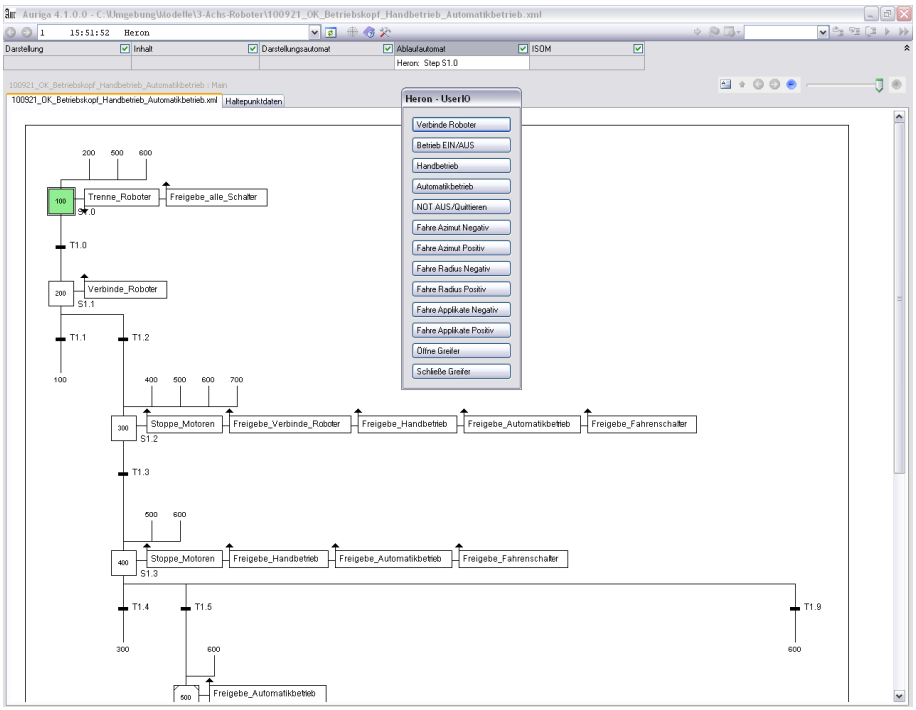
[11] Reis Robotics: *Grundlagen der Robotik*, Lehrmaterial, http://www.reisrobotics.de/Service+_+Support/Download/Grundlagen+der+Robotik.html.

[12] *Sensor*, http://de.wikipedia.org/wiki/Sensor.

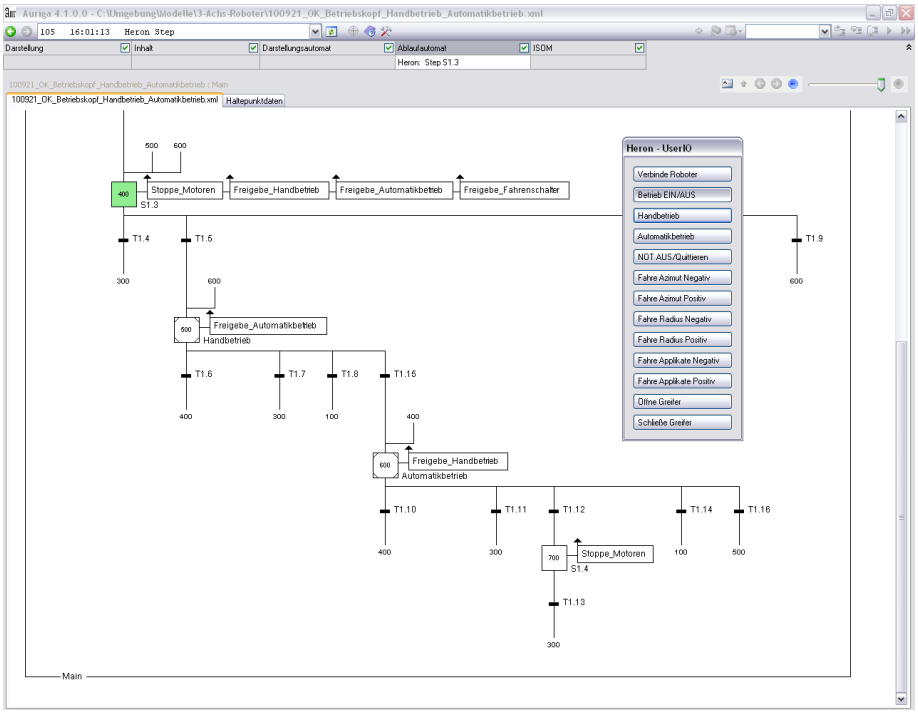
[13] *Robotersteuerung*, http://www.youtube.com/watch?v=3RRyroQj1aY.

Anhang

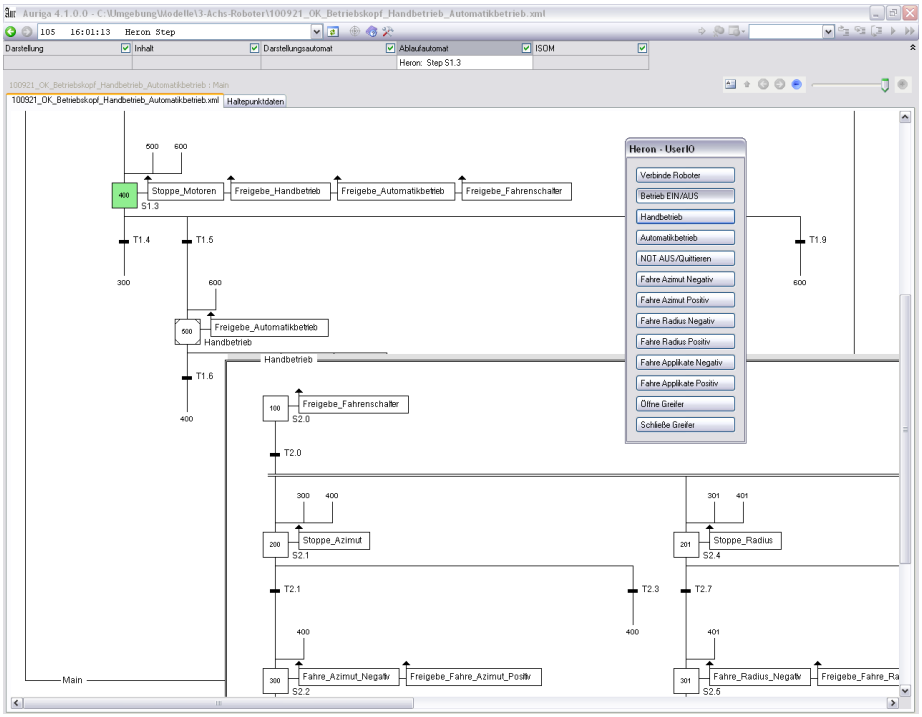
Anhang A1: Bildschirmfotos der Anwendungsoberfläche von Auriga



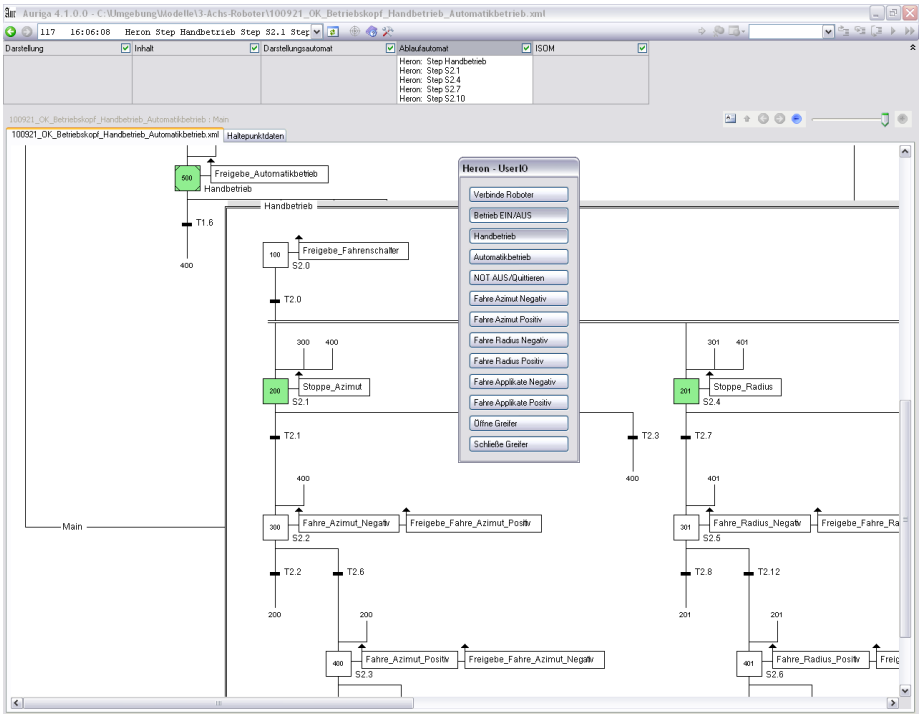
Anfangszustand von Auriga nach dem Aufruf des Modells



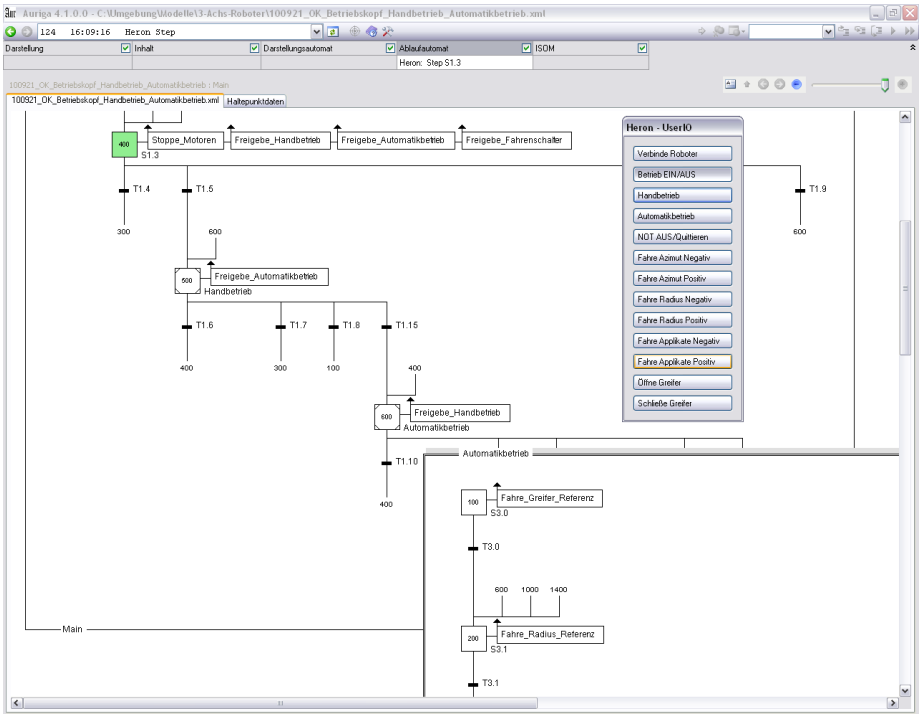
Hauptbetrieb ist eingeschaltet



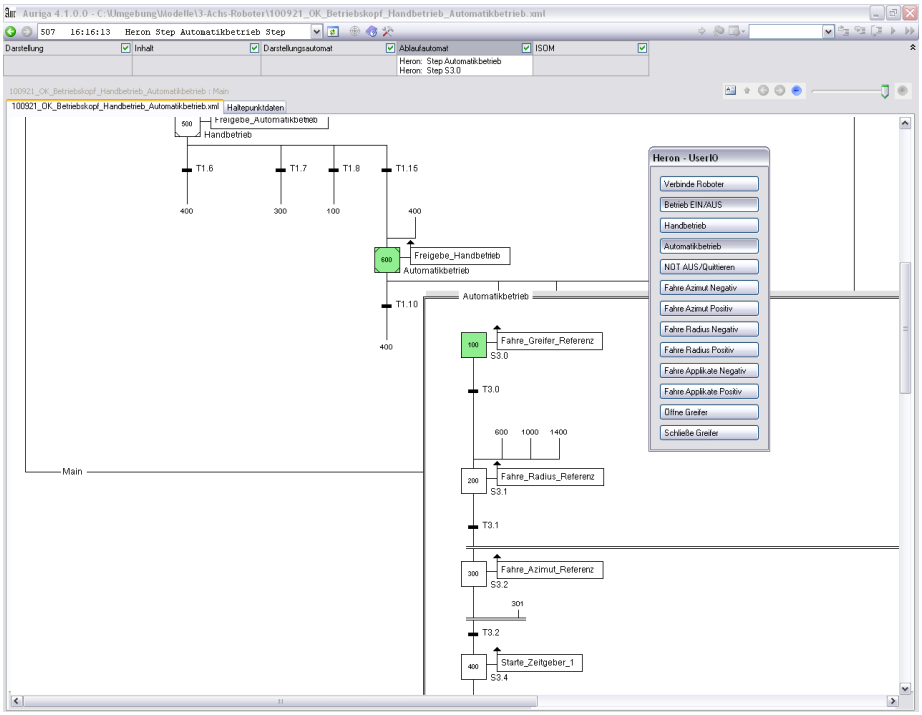
Öffnung der Anzeigeebene von Handbetrieb



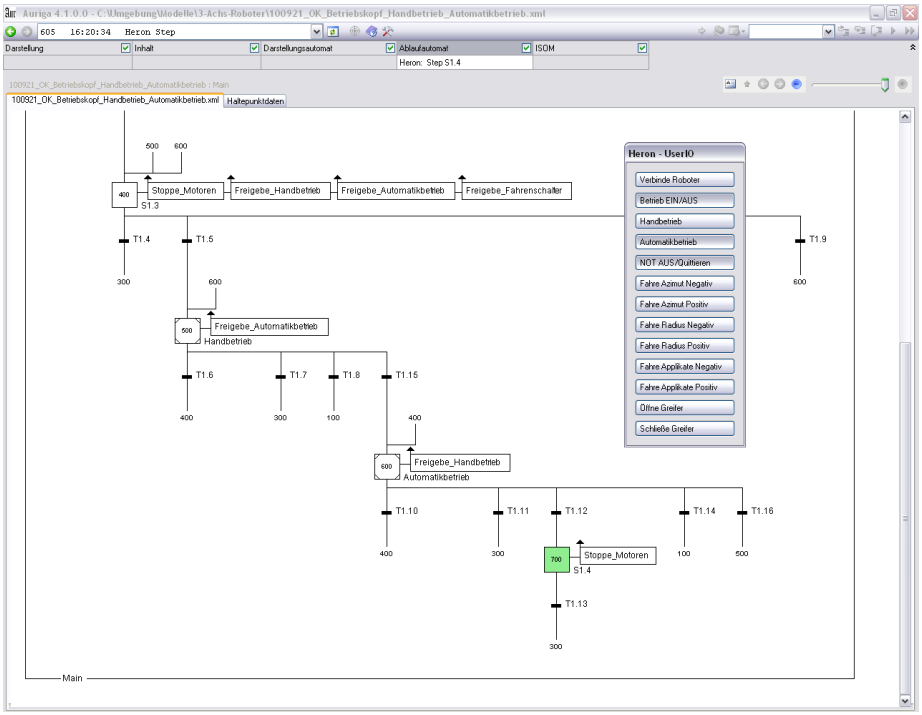
Handbetrieb ist aktiv



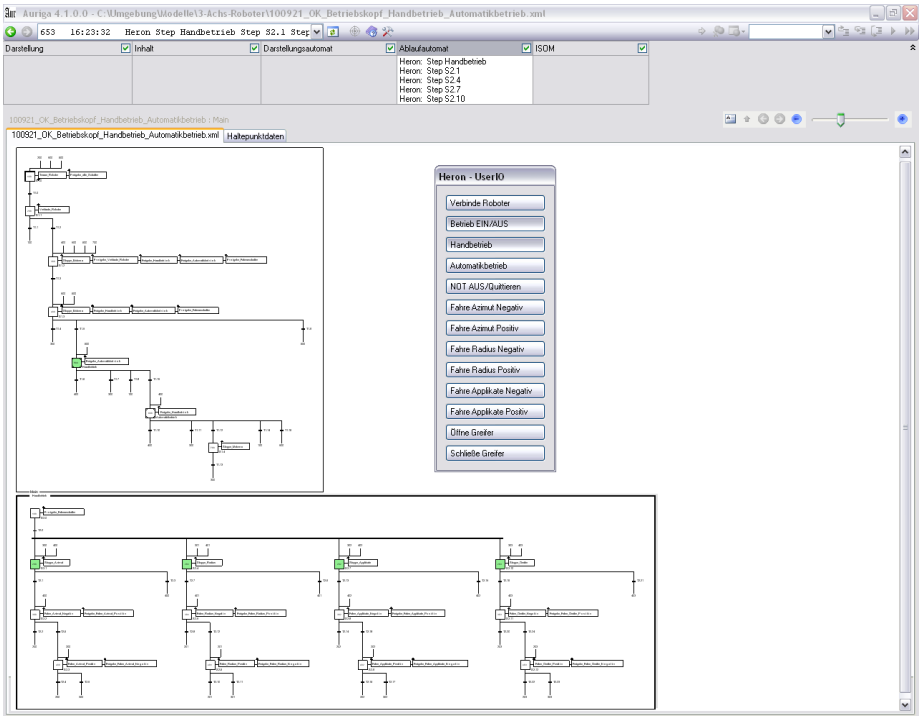
Öffnung der Anzeigeebene von Automatikbetrieb



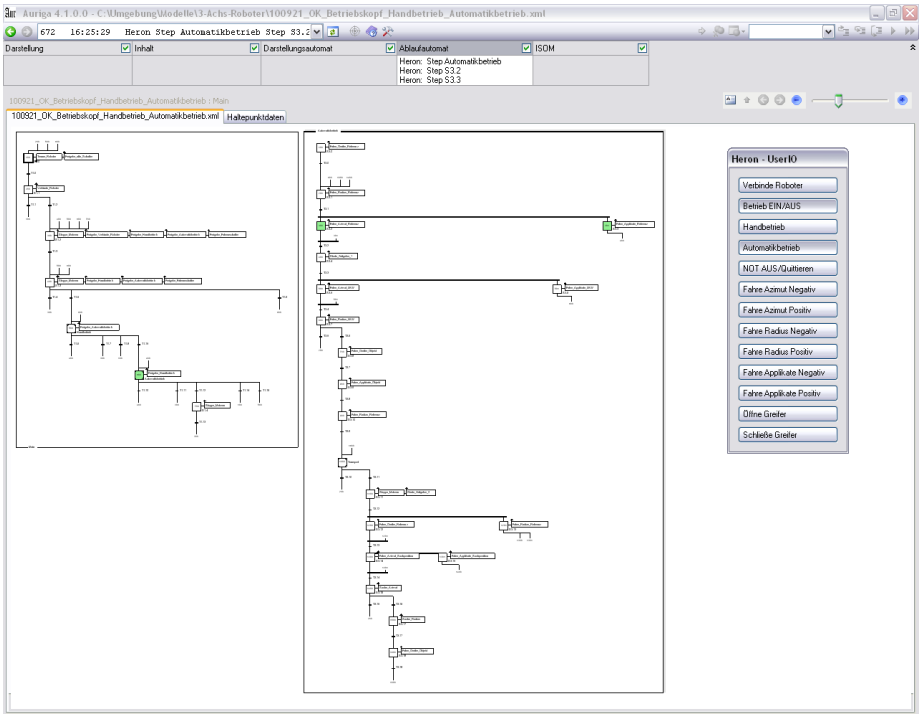
Automatikbetrieb ist aktiv



Schalter NOT AUS ist aktiviert



Gesamtblick von Betriebkopf und Handbetrieb



Gesamtblick von Betriebkopf und Automatikbetrieb

Anhang A2: Inhalt der beigefügten DVD

1) Diplomarbeit als PDF-Datei:
   Diplomarbeit_Bang_Giang_Nguyen.pdf
2) Quellcodes von *RobotModule* im Verzeichnis *FischertechnikRobot*
3) Petrinetz des Robotikexperiments:
   Petrinetz_Robotikexperiment.pdf
4) Steuerprogramm des Robotikexperiments in kanonische Beschreibungsform:
   Steuerprogramm_Robotikexperiment.xml
5) Video der Durchführung des Robotik-Experiments:
   Durchführung_Robotikexperiment.avi

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach § 25 (4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Ort, Datum	Unterschrift