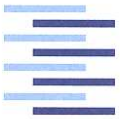

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Diplomarbeit

Alexander Oldenburger

Linux-Echtzeit-Programmierung mit dem Matlab Embedded Coder für die Webcam-basierte Bahnregelung eines mobilen Roboters

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

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

Alexander Oldenburger

Linux-Echtzeit-Programmierung mit dem Matlab Embedded Coder für die Webcam-basierte Bahnregelung eines mobilen Roboters

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. Thomas Holzhüter
Zweitgutachter: Prof. Dr.-Ing. Ulfert Meiners

Abgegeben am 24. September 2010

Alexander Oldenburger

Thema der Diplomarbeit

Linux-Echtzeit-Programmierung mit dem Matlab Embedded Coder für die Webcam-basierte Bahnregelung eines mobilen Roboters

Stichworte

Linux, Echtzeit, Vektorrotation, Prozessautomatisierung, Regelung, omnidirektionaler Roboter (Robotino), digitale Bildverarbeitung, Matlab/Simulink, Real-Time Workshop Embedded Coder, Simulation, Hardwaretreiber, RS-232, USB Webcam

Kurzzusammenfassung

Diese Arbeit behandelt die Anbindung einer USB Webcam an eine echtzeitfähige Ansteuerung eines mobilen Roboters sowie die Entwicklung mehrerer Regelprozesse dieses Systems. Ein konfiguriertes Linux Betriebssystem bildet die Grundlage für die einfache Treiberprogrammierung der Hardware und die Ausführung der unter Matlab/Simulink und dem RTW Embedded Coder entwickelten Regelalgorithmen. Weiterhin werden zugehörige Simulationsmodelle entworfen und mit den realen Abläufen verglichen..

Alexander Oldenburger

Title of the paper

Linux-realtime-programming with Matlab Embedded Coder for a webcam-based track-keeping control of a mobile robot

Keywords

Linux, realtime, vector rotation, process automation, control, omnidirectional robot (Robotino), imageprocessing, Matlab/Simulink, Real-Time Workshop Embedded Coder, simulation, hardware driver, RS-232, USB Webcam

Abstract

This paper decribes the linkage of a USB webcam to a real-time control system for a mobile robot and the development of several controllers for this system. For this purpose a configured Linux operating system forms the basis for easy hardware driver programming and for the execution of the control algorithms developed with Matlab/Simulink and the RTW Embedded Coder. Furthermore corresponding simulation models have been designed and compared to the real processes.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einleitung

Die erste vollmechanische Rechenmaschine Zuse Z1 wurde 1936 von Konrad Zuse erbaut. Basierend auf dünnen, bistabilen Metallflächen konnte die Zuse Z1 erstmals einfache duale Rechenalgorithmen durchführen und gilt damit als der Vorläufer des modernen Computers. Eigene Jahre darauf wurde der noch vollmechanisch laufende Apparat durch einen, auf dem binären Zahlensystem basierten, programmierbaren Digitalrechner den Zuse Z3 ersetzt. Angesichts dieser Innovation bezeichnet man den Zuse Z3 gegenwärtig als den ersten funktionsfähigen Computer der Geschichte. Der Ausbau der Computerentwicklung und die damit verbundenen Möglichkeiten nehmen seit dieser Zeit beinahe exponentiell zu. Die Umsatzfähigkeit der Unternehmen hängt deutlich von diesem Wachstum ab, denn die moderne Produktion ist ohne einen gegenwärtigen Computerstandard nicht vorstellbar. Gerade in der heutigen wirtschaftlichen Finanzkrise und der großen Anzahl der Marktkonkurrenz ist der Druck auf die Unternehmen so groß, dass fortwährende Erweiterungen und Flexibilität auf aktuelle Wünsche der Kunden eingehen zu können von existenzieller Wichtigkeit ist. Dabei spielen selbst kleinste Weiterentwicklungen, die den Produktionsprozess vorantreiben können, eine wesentliche Rolle.

In der Automatisierungstechnik werden mit *Schnittstellen*[1] Daten zwischen Computern und Maschinen ausgetauscht und dadurch Automatisierungsprozesse realisiert. Gebunden an die Entwicklung der digitalen Rechenmaschinen nimmt die Variation, die Effizienz und damit auch die Komplexität dieser Schnittstellen sowie der zugehörigen Software zu. Diese immerwährenden Innovationen müssen in der Industrie angepasst werden, sodass bei dem enormen Marktdruck die Konkurrenzfähigkeit bestehen bleibt. Doch es ist nicht einfach mit diesem Fortschritt standzuhalten. Beispielsweise wurde in den frühen 1960ern die *EIA-232*[2] Schnittstelle eingeführt und hinsichtlich der bequemen Handhabung und dem weit verbreiteten Einsatz bis zum Anfang des 21. Jahrhunderts in nahezu jedem Computer eingebaut und immer noch oft in der Automatisierungstechnik verwendet. Die geringe *Datenübertragungsrate*[3] der EIA-232 Schnittstelle macht komplexe Datenverarbeitung, die eine sehr hohe Datenrate erfordert, nicht mehr möglich. Darüber hinaus wird diese Schnittstelle in der modernen Hardware nicht mehr unterstützt. Diese Situation macht die Etablierung von leistungsfähigeren Übertragungsarten in der Automatisierungstechnik unumgänglich.

Parallel zu der Entwicklungsgeschichte des Computers wurde eine große Vielfalt von *Betriebssystemen*[4] hervorgerufen. Das meist verbreitete Betriebssystem wird heute im privaten Haushalt verwendet, das in erster Linie für Kommunikation und Multimedia eingesetzt wird. Aufgrund der relativ langen Einsatzphase der EIA-232, ist aus der Entwicklersicht in fast jedem Betriebssystem die Möglichkeit gegeben, die serielle Schnittstelle bequem

[1] Elektrischer Anschluss zum Austausch von Daten
[2] Ursprünglich RS-232, bezeichnet einen Standard für eine serielle Schnittstelle
[3] Übertragungsgeschwindigkeit von Daten
[4] Software zur Verwaltung der Hardware eines Computers

und einfach nutzen zu können. Für Automatisierungsprozesse ist jedoch das privat eingesetzte Betriebssystem nicht konstruktiv und wird daher immer weniger in diesem Bereich eingesetzt.

Aus diesem Hintergrund stellt sich die Frage, inwieweit kann in der Automatisierungstechnik moderne Hard- und Software eingesetzt werden, damit komplexe und zeitgemäße Steuer- und Regelabläufe realisiert werden können.

1.1 Motivation

Eine nahezu exponentiell wachsende Entwicklung der Hard- und Software zieht eine immer größer werdende Variation von leistungsfähigeren und komplexeren Systemen mit sich nach, die für das Wachstum der Automatisierung eine wichtige Funktion spielen. Diese Faktoren an sich verlangen schon ständig nach neuen Methoden moderne Technologie einsetzen zu können. Um dabei Zeit und Kosten zu sparen, sollten die vorgängigen Investitionen und Entwicklungen so weit wie möglich mit eingebunden werden.
Am Beispiel der schon inzwischen rückständigen EIA-232 Schnittstelle besteht die Motivation eine schnellere und zeitgemäße Schnittstelle wie den *Universal Serial Bus*[5] mit den gewohnten Mitteln nutzen und die auf dem aktuellen Markt vorhandene Geräte wie Webcams in die Steuer- und Regelabläufe einfach und kostengünstig einbinden zu können.

1.2 Aufgabenstellung

Ein mobiler, omnidirektionaler Roboter namens Robotino, der von den Unternehmen Festo Didactic für Schulungszwecke gebaut und an bildende Einrichtungen verkauft wird, besitzt derzeit eine spezielle Software „*RobotinoView*“, die es ermöglicht einige vorgefertigte Prozesse von Festo Didactic auszuführen. Darüber hinaus bietet diese eine Möglichkeit bedingte Steuer- und Regelabläufe des Robotinos selbst zu gestalten. Auch der Fachbereich der Automatisierungstechnik der Hochschule für Angewandte Wissenschaften besitzt für akademische Zwecke einige dieser Robotinos. Aktuell wird aber in dem Fachbereich die Software Matlab von MathWorks für Simulations- und Regelzwecke verwendet und daher wird RobotinoView leider nicht gebraucht. Im Bezug darauf wurden von Herrn Dipl.-Ing. Benjamin Tang und mir in einem Projekt die von Festo Didactic erstellen Funktionen modifiziert, sodass sie in Matlab integriert und weiterführende Projekte damit erstellt werden können. Matlab nutzt dabei das *Wireless Local Area Network*[6] zum Senden und Empfangen von Befehlen und Daten. Diese Methode ist allerdings wegen der WLAN Eigenschaften nicht immer fehlerlos und nicht *echtzeitfähig*[7]. Im Anschluss darauf wurde das Konzept der Steuerung des Robotinos von Herrn Dipl.-Ing. Elard Köhler in seiner Diplomarbeit „*Entwicklung einer kamera-basierten Bahnführungs-Regelung für einen omnidirektionalen*

[5] Aktueller Schnittstellenstandard für externe Hardware
[6] Drahtlose Datenübertragung zwischen zwei Systemen
[7] Programmabläufe in einer vordefinierten Zeit

Roboter mit Matlab xPC Target“ geändert. Ihm ist es gelungen mehrere, echtzeitfähige Bahnführungs-Regelungen mit Matlab und dem *xPC Target*[8] zu erzielen. Das xPC Target bedient dabei die EIA-232, die für die Steuerung der *Aktoren*[9] und *Sensoren*[10] zuständig ist, und die WLAN Schnittstelle, über die die Programmcodes geladen werden. Ein zweites System, mit einer an der Decke angebrachten Kamera, sorgt für die Ortung und Koordination des Roboters. Die fehlende USB Unterstützung des xPC Targets macht allerdings die Anbindung von *USB-Devices*[11] in diesem Gefüge nicht möglich.

Die Kernaufgabe dieser Diplomarbeit bezieht sich also auf die Entwicklung einer echtzeitfähigen Regelung des Robotinos mit Matlab/Simulink unter Gebrauch der USB Schnittstelle. Am Beispiel des Vorgängermodells soll ein an der Decke angebrachtes Light Emitting Diode Muster von einer am Robotino angeschlossener Webcam erkannt und eine zugehörige koordinierte Regelung realisiert werden. Im Hinblick darauf wird eine geeignete Systemplattform ausgewählt und die Kernsteuerung des Robotinos neu konstruiert.

1.3 Zielsetzung

Die Zielsetzung dieser Arbeit ist die Optimierung der bisher vorhandenen Bedienarten des Robotinos, sodass mit Matlab/Simulink eine echtzeitfähige, koordinierte Regelung mit einer an dem Robotino angeschlossener USB Webcam realisiert werden kann. Am Ende dieser Arbeit werden daher die nachfolgenden Punkte erreicht oder zumindest untersucht worden sein:

- Untersuchung und Auflistung der Hardware des Robotinos
- Anwendungsmöglichkeiten von passenden Betriebssystemen
- Eigenschaften einer EIA-232 Schnittstelle und deren Ansteuerung der Aktoren und Sensoren des Robotinos
- Verbindung zweier Betriebssysteme unter Verwendung von Matlab/simulink
- Beschreibung der Programmiermethoden
- WLAN Einbindung zur Fernsteuerung
- Ansteuerung und Eigenschaften der USB Webcam
- Entwurf eines LED Musters zur Koordination des Robotinos
- Bildverarbeitung der geschossenen Bilder zur Koordination des Robotinos
- Echtzeitverhalten der Steuer- und Regelverläufe

[8] Matlab/Simulink Toolbox für eine echtzeitfähige Systemansteuerung
[9] Hier: Die Antriebsmotoren des Robotinos
[10] Technische Messfühler für physikalische Eigenschaften
[11] Ein USB Endgerät, das an einen Computer angeschlossen wird

- Realisierung von mehreren Regelabläufen und die Untersuchung deren Verhalten

Die Verwirklichung der aufgelisteten Punkte sowie noch weitergehende Aspekte und Ziele stellen das Wesentliche und gleichzeitig die Schwierigkeit dieser Arbeit dar.

1.4 Vorgehensweise

Nach einer Untersuchung der Robotinohardware und einer Einarbeitung in das Vorgängermodell wird entschieden, welches Betriebssystem und welche USB Webcam für die anstehenden Aufgaben geeignet ist. Anhand des kompatiblen Betriebssystems wird recherchiert, ob eine USB Unterstützung gegeben ist und wie diese von Matlab/Simulink angesprochen werden kann. Beim positiven Ergebnis wird das neue Betriebssystem installiert, die Kernsteuerung der Aktoren und Sensoren des Robotinos über die vorhandene EIA-232 Schnittstelle unter Matlab/Simulink realisiert und mit dem neuen System verknüpft. Darüber hinaus wird die Ansteuerung der USB Webcam in diesem Entwurf mit enthalten sein und die Echtzeitfähigkeit dabei berücksichtigt bleiben. Eine Fernsteuerung über die vorhandene WLAN Schnittstelle, zum Automatisieren der folgenden Prozesse der Steuerung oder Regelung, wird erstellt. Nach der Fertigstellung der gesamten Steuerbarkeit des Robotinos wird im Sinne der Evaluierung eine Lampe an der Decke angebracht und eine einfache Regelung des Roboters unter dieser erstellt. Zur Analyse des Fahrverhaltens wird ein Verfahren geschaffen, welches die gefahrenen Daten speichert. Die Daten werden anschließend mit einem Simulationsmodell des Robotinos verglichen. Mit den Methoden der digitalen Bildverarbeitung wird diese Regelung optimiert und ebenfalls mit den zugehörigen simulierten Daten verglichen. Letztendlich wird die Lampe durch ein LED Muster ersetzt, wodurch ebenfalls mit der digitalen Bildverarbeitung die Position und zusätzlich die Fahrrichtung des Roboters berechnet werden. Dies wird zu anspruchsvolleren Regelalgorithmen führen, die weiterhin analysiert und mit entsprechenden Simulationsmodellen verglichen werden.

1.5 Beschreibung der Hauptkapitel

Die **Einleitung** stellt allgemein das Thema vor und weist dabei auf die Motivation, die Aufgabe und auf die Zielsetzung dieser Diplomarbeit. Des Weiteren wird in diesem Abschnitt kurz die Vorgehensweise und die Zusammenfassung der Hauptkapitel präsentiert.

Der Textabschnitt **Hintergrund** stellt die Hochschule für Angewandte Wissenschaften vor und verbalisiert die Zielsetzung des Fachbereichs für Automatisierung. Außerdem wird MathWorks und Festo Dedactic sowie fachübergreifend, parallele Einsatzmöglichgeiten mit dem Robotino vorgestellt.

Das Kapitel **Technischer Hintergrund** stellt die Hardwarekomponenten des Robotinos dar und beschreibt in kurzen Zügen die Funktionsweise der relevanten Schnittstellen. Wieterhin wird zweckmäßige Software vorgestellt, die im Bezug auf die Aufgabenstellung hinterfragt

wird.

Unter **Systemaufbau** wird das Grundgerüst für die Realisierung der Zielsetzungen aufgebaut. Hierbei werden alle benötigten Funktionen und Komponenten für den Einsatz der anstehenden Regelungen konfiguriert und erläutert. Zur Erprobung des Systems wird zum Schluss ein Testmodell konstruiert und geprüft.

In dem Abschnitt **Realisierung** werden letztendlich die letzten Komponenten für die Regelfahrten *justiert*[12] sowie einige realisierungsabhängige Zusammenhänge untersucht. Nach einem erfolgreichen Systemaufbau und den vorgenommen Einstellungen werden die geplanten Regelprozesse mit den zugehörigen Simulationen entwickelt und in Hinsicht auf ihr Regelverhalten analysiert.

Im letzten Kapitel **Abschluss** wird eine kurze Zusammenfassung der ganzen Diplomarbeit vorgestellt und ein abschließendes Fazit zu dem Arbeitsverlauf gegeben. Zusätzlich werden weiterführende Einsatzmöglichkeiten und Optimierungen präsentiert sowie eine Danksagung an helfende Personen übermittelt.

[12] Einstellung der technischen Komponenten

2 Hintergrund

In diesem Kapitel wird die Hochschule für Angewandte Wissenschaften und die Zielsetzung des Fachbereichs Automatisierungstechnik vorgestellt. Des Weiteren wird ein Einblick auf fachübergreifend ähnlich laufende Projekte sowie MathWorks, Festo Didactic und den Robotino selbst gezeigt.

2.1 Kurzporträt Hochschule für Angewandte Wissenschaften

Die Hochschule für Angewandte Wissenschaften hat rund 12000 Studierende in über 50 technischen, wirtschaftlichen, sozialen, gestalterischen und informationsbezogenen Studiengängen und rund 336 Professorinnen und Professoren. Sie zählt damit zu der zweitgrößten Hochschule Hamburgs und einer der größten ihrer Art in Deutschland.

Das Department Informations- und Elektrotechnik mit knapp 1000 Studenten und 40 Professoren bietet zurzeit neben der Fachrichtung Automatisierungstechnik noch Informationstechnik und Kommunikationstechnik an. In den letzten Jahren wurde das bis dahin geltende Studiensystem mit einem Diplomabschluss in ein neues, zweistufiges Bachelor-Master System geändert. Dementsprechend findet im Fachbereich Automatisierungstechnik zum ersten Mal im Wintersemester 2010/2011 ein Masterstudiengang statt.

Abbildung 2.1: Hochschule für Angewandte Wissenschaften (HAW , Zugriff: 19.07.2010)

2.1.1 Zielsetzung der HAW

Der Robotino wird nicht nur im Fachbereich Automatisierungstechnik für Schulungen eingesetzt, sondern stellt auch für andere Bereiche eine zweckdienliche Basis für Projekte dar. Momentan wird in der Informationstechnik an einer Sprachsteuerung gearbeitet, dabei empfängt eine externe Anordnung die akustischen Signale und wandelt sie in Fahrbefehle um. Am Beispiel der modernen Navigation werden ähnliche Versuche der Ortung durchgeführt. Im Fachbereich Kommunikationstechnik versuchen die Studenten neue Aktoren und Sensoren zu entwerfen, die in der Lage sind Gegenstände zu heben und zu transportieren. Aufgrund des neuen Studiensystems und dem erstmals startenden Masterstudiengang im Fachbereich Automatisierungstechnik, müssen die bisherigen Projekte erweitert werden. In dieser Diplomarbeit wird ein neues Verfahren zur Steuerung und Regelung des Robotinos entwickelt, an dem eine USB Webcam angeschlossen ist. Dieses Verfahren soll dann künftig von Studenten in weiterführenden Projekten des Masterstudiengangs eingesetzt werden.

2.2 Kurzportät Festo Didactic

Festo ist ein weltweit führendes Unternehmen in der Automatisierungstechnik sowie der technischen Aus- und Weiterbildung. Im Vordergrund steht für sie die maximale Produk-

tivität und Wettbewerbsfähigkeit von Kunden in der Fabrik- und Prozessautomatisierung. Dabei umfasst Festo Didactic mit seiner Erfahrung und einer 35-jährigen Unternehmensgeschichte das Angebot der Bildungsausrüstungen für Aus- und Weiterbildungseinrichtungen im gesamten Spektrum der Fertigungs- und Prozessautomatisierung. Unter anderem wurde der Robotino (siehe Abbildung 2.2) entwickelt und von der HAW erworben.

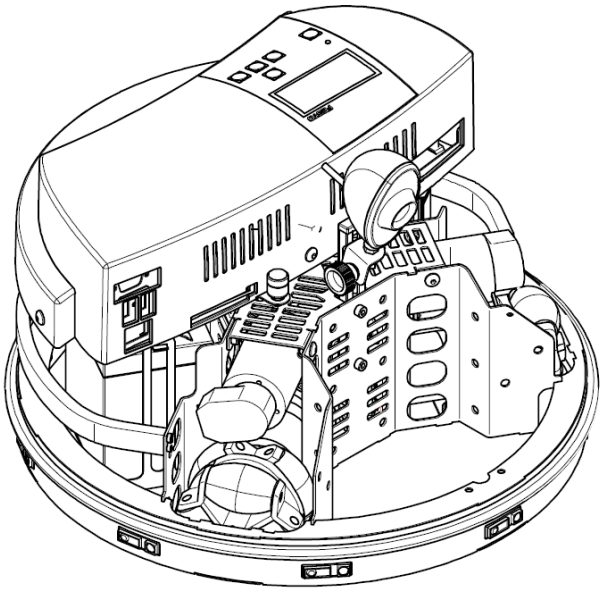

Abbildung 2.2: Robotino von Festo Didactic (Robotino)

2.3 Kurzportät The MathWorks

The MathWorks ist mit mehr als 2000 Mitarbeitern seit 1984 der weltweit führende Anbieter von Software für Technical Computing und Model-Based Design. Matlab von MathWorks ist eine Programmierumgebung zur Lösung von mathematischen Problemen, die im Vordergrund für numerische Berechnungen mit Matrizen ausgelegt ist. Darüber hinaus bietet Matlab Algorithmen zur Analyse und Visualisierung von Daten. Matlab/Simulink ist eine grafische Oberfläche zur Modellierung von dynamischen Systemen in Programmblöcken. Die verschiedenen kontinuierlichen und diskreten, zur Verfügung stehenden Blöcke bieten die Möglichkeit, unter Matlab auch andere Programmiersprachen in den grafischen Datenfluss einzubinden und damit Simulationen von physischen Systemen nachzubilden oder automatisierte Prozessabläufe zu erstellen. Dieses Programm ist weltweit für Bildungszwecke in Universitäten etabliert und wird von Ingenieuren und Wissenschaftlern in vielen Einsatzgebieten für Optimierungen und Innovationen in der Wirtschaft eingesetzt.

Aufgrund der weltweiten Etablierung dieses Programms in der Wirtschaft und der Universitäten, wird in dieser Diplomarbeit unter anderem Matlab/Simulink zur Steuerung, Regelung und Simulation des Robotinos eingesetzt. Zusätzlich kann dabei teilweise das Beispiel der Vorgängermodelle aufgegriffen und modifiziert werden.

3 Technischer Hintergrund

In diesem Kapitel werden die einzelnen Bestandteile des Robotinos und deren Bedeutung für die Dynamik des Roboters gezeigt. Weiterhin wird die Funktionsweise der wesentlichen Schnittstellen erläutert, auf die Aufgabe bezogene Software präsentiert und deren Vor- und Nachteile für den Einsatz der Robotinosteuerung abgewogen.

3.1 Robotino

Im Fachgebiet Automatisierung übernehmen Roboter aufwendige und präzise Aufgaben in der industriellen Herstellung. Die Hochschulen verwenden kleine Modelle wie den Robotino, um Studenten einen Einblick in diese Welt geben zu können. Der Robotino ist ein mobiler, omnidirektionaler Roboter von der Firma Festo Didactic, der speziell für diese Ausbildungszwecke hergestellt wird. Mit einer Vielzahl von angeschlossenen Hardwarekomponenten bietet er die ideale Basis dieser Diplomarbeit.

Der Robotino selbst besteht hauptsächlich aus einem *Chassis*[13] und einer Kommandobrücke. Zusammengesetzt ist er etwa 21 cm hoch, hat einen Durchmesser von 36 cm und wiegt ca. 12 kg (für genauere Daten siehe Tabelle 3.1). Die Kommandobrücke beinhaltet einen PC/104 und ein Board mit Atmel Mikrokontrollern. Mit Hilfe der EIA-232 Schnittstelle kommuniziert der PC/104 mit dem Mikrokontroller Board und bildet somit die gesamte Steuereinheit der Aktoren und Sensoren des Robotinos. Zusätzlich stellt der PC/104 zwei USB Ports, eine VGA Schnittstelle, ein Compact Flash Lesegerät und einen LAN Anschluss zur Verfügung, deren Anschlüsse auf die Kommandobrücke herausgeführt und verwendet werden können. Hinzu kommt noch eine Steckerleiste mit zwei Relais, acht analogen Eingängen und acht digitalen Ein- und Ausgängen (siehe Abbildung 3.1).

[13] Fahrgestell des Robotinos

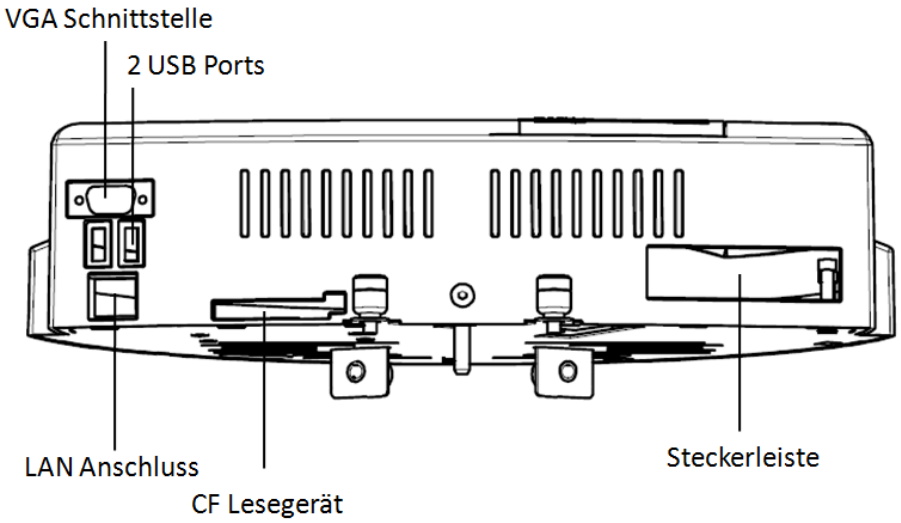

Abbildung 3.1: Kommandobrücke vom Robotino (Robotino)

Das Chassis formt das Trägergerüst, auf dem sich der Stoßbumper, zwei 24 V Bleiakkus, drei omnidirektionale Räder mit zugehörigem Getriebe und Gleichstrommotoren, neun Abstandssensoren sowie die Kommandobrücke selbst befinden. Zusätzlich gibt es dort Platz für weitere anschließbare Geräte, wie einer Webcam oder eines Access Points (siehe Abbildung 3.2).

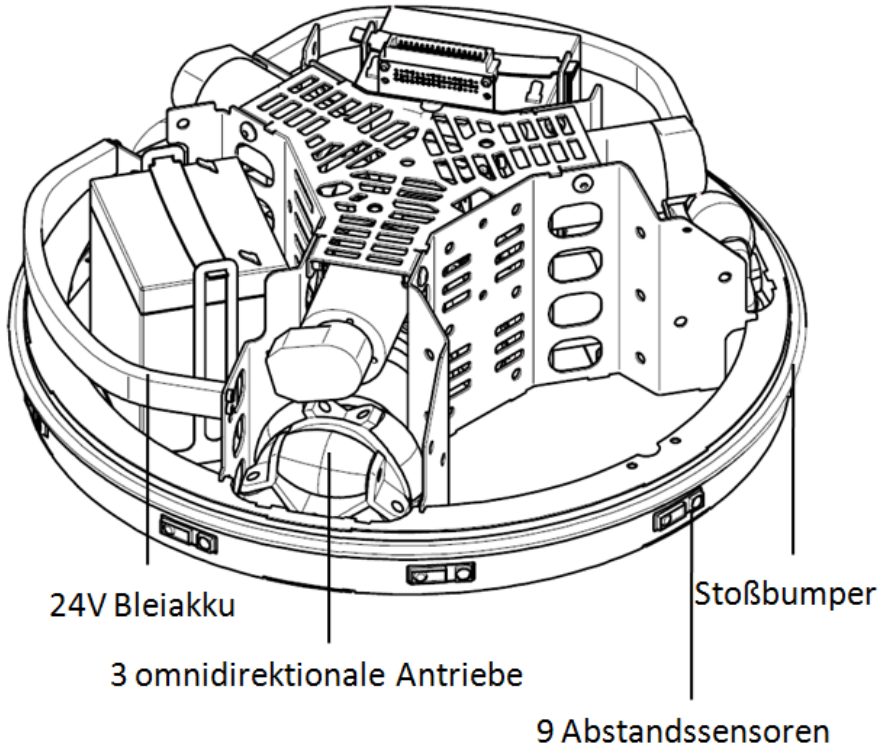

Abbildung 3.2: Chassis vom Robotino (Robotino)

In der folgenden Tabelle sind die Maße des Robotinos noch mal genauer aufgelistet:

Objekt	Länge	Betrag
Räder	Durchmesser	80 mm
	Tiefe	60 mm
Kommandobrücke	Höhe	85 mm
	Breite	300 mm
	Tiefe	190 mm
Robotino	Höhe (mit Kommandobrücke)	210 mm
	Durchmesser	360 mm
	Roboter-Mittelpunkt bis Mitte eines Rades	135 mm

Tabelle 3.1: Abmessungen vom Robotino (Robotino)

3.1.1 PC/104

Der PC/104 ist ein industrieller Computer mit einer Leiterplattengröße von 90,17 mm x 95,89 mm und kann zusammengesteckt ein komplexes Rechensystem bilden. Wegen der Größe und der zweckgemäßen Anwendbarkeit werden sie vermehrt in der Elektroindustrie eingesetzt. Mit einer AMD Prozessorleistung von 500 Mhz, einem Arbeitsspeicher von bis zu 1 GB DDR und seinen zwei EIA-232, zwei USB, einer VGA, einer LAN, einer ISA und einer PCI Schnittstelle bildet er das Herzstück des Robotinos. Der PC/104 übernimmt mit einer seiner EIA-232 Schnittstelle die Ansteuerung der Aktoren und Sensoren des Robotinos, in dem er vordefinierte Bytesequenzen an ein Stuerboard mit Atmel Mikroprozessoren sendet und empfängt. An ein zusätzlich angeschlossenes Compact Flash Lesemodul an dem PC/104 kann in diesem Fall eine 4 GB große Speicherkarte eingesetzt werden. Auf diese Speicherkarte können Betriebssysteme zur Steuerung des Robotinos installiert und über das BIOS des PC/104 *gebootet*[14] werden. Des Weiteren ist auf dem Board ein VGA Anschnluss vorhanden, mit dem ein analoger Computerbildschirm angeschlossen und dadurch visuell das Betriebssystem bedient werden kann.

3.1.2 Abstandskontrolle

Es befinden sich neun Infrarot-Abstandssensoren auf dem Chassis des Robotinos, die in einem Winkel von 40° angebracht sind (siehe Abbildung 3.3). Die Sensoren sind in der Lage die Entfernung von naheliegenden Gegenständen in einem Areal von 40 mm bis 300 mm zuverlässig zu messen.

[14] Systemstart

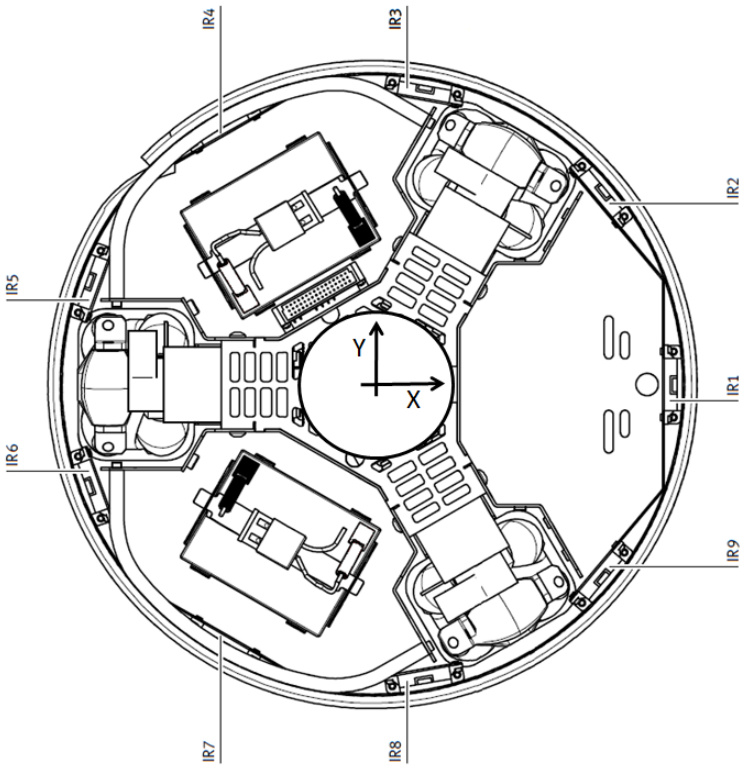

Abbildung 3.3: Infrarot-Abstandssensoren (Robotino)

Zusätzlich zu den Abstandsensoren wurde, wie in der Abbildung 3.2 sichtbar, sicherheitstechnisch noch ein *Stoßbumper*[15] angebracht. Der schwarze Gummischlauch beinhaltet zwei Kontakte, die bei einer Kollision mit einem Gegenstand aneinander gedrückt werden und dadurch ein digitales Signal erzeugen. Dieses Signal soll zweckgemäß den Robotino abschalten, um entstehende Schäden zu vermeiden.

3.1.3 Steckerleiste

Die an dem Robotino angebrachte 20-polige Ein- und Ausgangssteckerleiste wird von zwei 24V Bleiakkus versorgt (siehe Abbildung 3.1) und dient zum Anschluss von zusätzlicher Hardware. Sie verfügt über zwei Relais, acht analoge Eingänge und acht digitale Ein- und Ausgänge (siehe Abbildung 3.4).

[15] Schutzkomponente gegen Stoßschäden

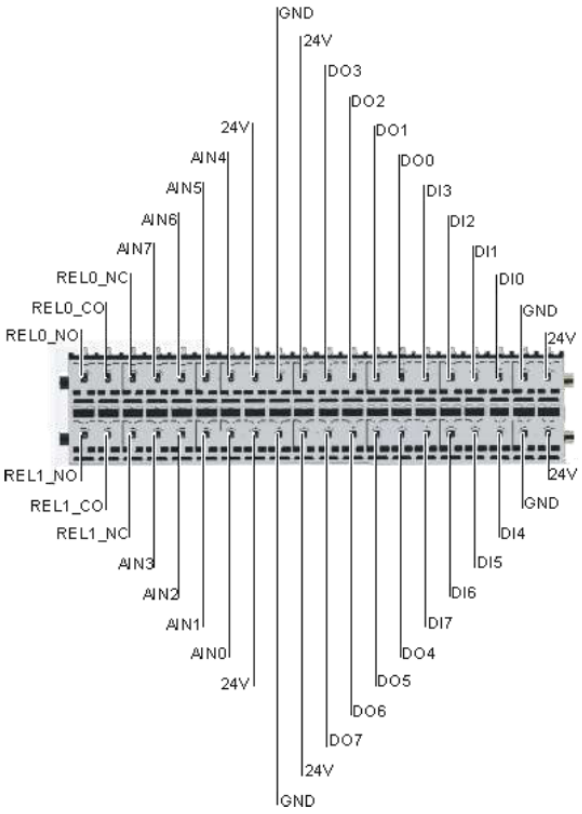

Abbildung 3.4: Steckerleiste am Robotino (Robotino)

2 Relais (REL0 und REL1) 24-30 V maximal 500 mA (achtung, nicht Kurzschlusssicher). Die Relais können als Öffner (NC), Schließer (NO) oder Wechsler (CO) verwendet werden

8 analoge Eingänge (AIN0 bis AIN7) U_{in}=0 bis 10 V; I_{min}>1 mA; 50 Hz

8 digitale Ausgänge (DO0 bis DO7) $U_{A_{nom}}$=24 V; $I_{A_{max}}$=0,3 A (kurzschlusssicher)

8 digitale Eingänge (DI0 bis DI7) U_{in}=0 bzw. 24 V; I_{min}>1 mA

3.1.4 Fahrdynamik

Die Dynamic des Robotinos wird von drei, voneinander unabhängigen, omnidirektionalen Rädern bewerkstelligt. Sie sind jeweils in 120° zueinander am Chassis des Robotinos angebracht. Mit einem Durchmesser von 80 mm wird eine Tragfähigkeit von 40 kg gewährleistet. Dies wurde in der Abbildung 3.5 nochmal deutlicher veranschaulicht. Jedes dieser Räder verfügt über sechs Rollen, die von der Hauptachse aus gesehen, eine gleichzeitige Bewegung in die X- und Y-Richtung möglich machen. Der Antrieb selbst besteht im Einzelnen aus einem Gleichstrommotor (für Leistungsdaten siehe Tabelle 3.2), einem Getriebe mit einem Übersetzungsverhältnis von 16:1, einem Zahnriemen, einem *Inkrementalgeber*[16] und den

[16]Sensor zur Erfassung der Lageänderung

bereits beschriebenen Rädern. Die Inkrementalgeber liefern die reale Motorgeschwindigkeit zurück, sodass mit einer PID-Regelung die beabsichtigte Fahrgeschwindigkeit gehalten werden kann.

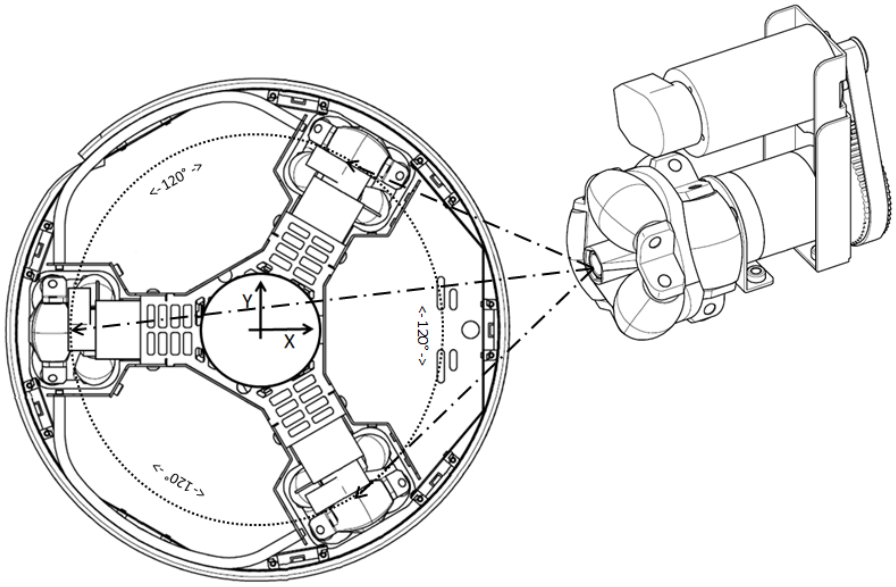

Abbildung 3.5: Antriebsauslegung des Robotinos (Robotino)

Die folgende Tabelle beinhaltet die wichtigsten Leistungsmerkmale der drei Gleichstrommotoren, die den Robotino antreiben:

Gleichstrommotor (GR 42x25)	**Wert**	**Einheit**
Nennspannung	24	VDC
Nenndrehzahl	3600	rpm
Nenndrehmoment	3,8	Ncm
Nennstrom	0,9	A
Anlaufmoment	20	Ncm
Anlaufstrom	4	A
Leerlaufdrehzahl	4200	rpm
Leerlaufstrom	0,17	A
Entmagnetisierstrom	6,5	A
Trägheitsmoment	71	gcm^2
Motorgewicht	390	g

Tabelle 3.2: Leistungsdaten eines Gleichstrommotors (Robotino)

3.1.5 Serielle Schnittstelle

Für die Ansteuerung der Gleichstrommotoren und den Empfang der Sensordaten ist der PC/104 und das Steuerboard mit den Atmel Mikrokontrollern (I/O Board) verantwortlich.

Das I/O Board wurde vom Hersteller so vordefiniert, dass es von dem PC/104 einen 47 Datenbyteblock erwarten, der unter anderem Befehle zu den Geschwindigkeiten der Motoren enthält. Bei einem Erfolgreichen und Vollständigen Empfang der Daten sendet das I/O Board einen 101 Datenbyteblock als Antwort an den PC/104 zurück. Darunter befinden sich Angaben zu den aktuellen Sensordaten, beispielsweise über die Inkrementalgeber gemessene Geschwindigkeit des Robotinos. Die Bedeutung der einzelnen Bytes der beiden Blöcke ist in der Tabelle 3.3 und der Tabelle 3.4 aufgelistet. Die Steuerung wurde insgesamt für vier Motoren vorgesehen, obwohl der Robotino nur drei Antriebsmotoren besitzt. Deshalb wird die Ansteuerung des vierten Motors außer acht gelassen. Eine fehlerfreie Übertragung wird durch drei Startbytes „R, E, C“ und drei Stoppbytes „r, e, c“ der beiden Blöcke gewährleistet. Wenn einer der Start- oder Stoppbytes nicht in den Blöcken enthalten ist, wird davon ausgegangen, dass die empfangenen oder gesendeten Daten nicht korrekt sind.

3.1.5.1 Transferblöcke des I/O Boards

In der nachfolgenden Tabelle sind die 47 Datenbytes aufgelistet, die zum I/O Board vom PC/104 gesendet werden müssen, um die jeweiligen Komponenten anzusteuern:

Byte	Bit	Beschreibung
0	0-7	Start-Byte 0, das Zeichen „R“
1	0-7	Start-Byte 1, das Zeichen „E“
2	0-7	Start-Byte 1, das Zeichen „C“
3	0	Notfall Stopp, 0: Stopp, 1: OK
	1	Power, 0: an, 1: aus
	2	Master LED, 0: aus, 1: an
4	0	Bremse von Motor 1, 0: Bremse aktiviert, 1: Bremse deaktiviert
	1	Digitaler Ausgang 0
	2	Digitaler Ausgang 1
	3	Digitaler Ausgang 2
	4	Digitaler Ausgang 3
	5	Relais 0
	6	LED, 0: aus, 1: an
5	0	Modus (Ansteuerung Motor 0), 0: Geschwindigkeit, 1: Position
	1	Drehrichtung Motor 0, 0: negativ, 1: positiv
	2	Rücksetzen der Motorlaufzeit
	3	Rücksetzen der Motorposition (bzw. des Inkrementalgeberwertes)
6	0-7	Geschwindigkeit Motor 0 (0-255)
7	0-7	Motorposition Bits 0-7
8	0-7	Motorposition Bits 8-15
9	0-7	Motorposition Bits 16-23
10	0-7	Motorposition Bits 24-31
11	0-7	K_p Motor 0 (0xff für Default-Werte)

12	0-7	K_i Motor 0 (0xff für Default-Werte)
13	0-7	K_d Motor 0 (0xff für Default-Werte)
14-23		Motor 1: wie Byte 4-13; Relais 1 kann über Byte 14 geschaltet werden
24-33		Motor 2: wie Bytes 4-13
34-43		Motor 3 (optional): wie Bytes 4-13
44	0-7	Stopp-Byte 0, das Zeichen „r“
45	0-7	Stopp-Byte 1, das Zeichen „e“
46	0-7	Stopp-Byte 1, das Zeichen „c“

Tabelle 3.3: Datenbyteblock zur Steuerung der Aktoren (Köhler, 2010)

Die Definition der 101 Bytes der Antwort, die vom I/O Board an den PC/104 zurückgesendet werden, wird in der nachfolgenden Tabelle genauer beschrieben. Generell besteht die Antwort aus den gemessenen Daten aller Sensoren des Robotinos.

Byte	**Bit**	**Beschreibung**
0	0-7	Start-Byte 0, das Zeichen „R“
1	0-7	Start-Byte 1, das Zeichen „E“
2	0-7	Start-Byte 1, das Zeichen „C“
3	0-7	Master ADU 0 Bits 2-10
4	0-7	Master ADU 1 Bits 2-10
5	0-7	Master ADU 2 Bits 2-10
6	0-7	Master ADU 3 Bits 2-10
7	0-7	Master ADU 4 Bits 2-10
8	0-7	Master ADU 5 Bits 2-10
9	0-7	Master ADU 6 Bits 2-10
10	0-7	Master ADU 7 Bits 2-10
11	0-1	Master ADU 0 Bits 0-1
	2-3	Master ADU 1 Bits 0-1
	4-5	Master ADU 2 Bits 0-1
	6-7	Master ADU 3 Bits 0-1
12	0-1	Master ADU 4 Bits 0-1
	2-3	Master ADU 5 Bits 0-1
	4-5	Master ADU 6 Bits 0-1
	6-7	Master ADU 7 Bits 0-1
13	0-7	Master time
14	0-7	Slave ADU 0 Bits 2-10
15	0-7	Slave ADU 1 Bits 2-10
16	0-7	Slave ADU 2 Bits 2-10
17	0-7	Slave ADU 3 Bits 2-10
18	0-7	Slave ADU 4 Bits 2-10
19	0-7	Slave ADU 5 Bits 2-10
20	0-7	Slave ADU 6 Bits 2-10

21	0-7	Slave ADU 7 Bits 2-10
22	0-1	Slave ADU 0 Bits 0-1
	2-3	Slave ADU 1 Bits 0-1
	4-5	Slave ADU 2 Bits 0-1
	6-7	Slave ADU 3 Bits 0-1
23	0-1	Slave ADU 4 Bits 0-1
	2-3	Slave ADU 5 Bits 0-1
	4-5	Slave ADU 6 Bits 0-1
	6-7	Slave ADU 7 Bits 0-1
24	0	Drehrichtung Motor 0 Bits 0-7
25	0-7	Geschwindigkeit Motor 0
26	0-7	Position Motor 0 Bits 0-7
27	0-7	Position Motor 0 Bits 8-15
28	0-7	Position Motor 0 Bits 16-23
29	0-7	Position Motor 0 Bits 24-31
30	0	Digitaler Eingang 0
	1	Digitaler Eingang 1
	2	Digitaler Eingang 2
	3	Digitaler Eingang 3
	4	Bumper
31	0-7	Motorlaufzeit Bits 0-7
32	0-7	Motorlaufzeit Bits 8-15
33	0-7	Motorlaufzeit Bits 16-23
34	0-7	Motorlaufzeit Bits 24-31
35-55		Motor 1: wie Bytes 14-34
56-76		Motor 2: wie Bytes 14-34
77-97		Motor 3 (optional): wie Bytes 14-34
98	0-7	Stopp-Byte 0, das Zeichen „r“
99	0-7	Stopp-Byte 1, das Zeichen „e“
100	0-7	Stopp-Byte 1, das Zeichen „c“

Tabelle 3.4: Datenbyteblock der Antwort von den Sensoren (Köhler, 2010)

3.1.5.2 Datenübertragung

Der Datenverkehr zwischen dem I/O Board und dem PC/104 wird über die EIA-232 Schnittstelle vorgenommen. Die Signale werden dabei im *Vollduplex-Betrieb*[17], asynchron und *seriell*[18] über ein Verbindungskabel gesendet und empfangen. Der asynchrone Transfer agiert dabei zeitunabhängig, sodass die Informationen nicht zu einem vereinbarten, sondern zu einem beliebigen Zeitpunkt versendet werden können. Damit gehört es zur Aufgabe des

[17] Senden und Empfangen von Daten zur gleichen Zeit
[18] Hintereinander

Empfängers den Beginn und das Ende des Dateneingangs festzustellen. Wegen den vielseitigen Eigenschaften der Schnittstelle, ist der Anschluss mit neun Pins ausgestattet (siehe Abbildung 3.6). Für die Ansteuerung des Robotinos sind jedoch die Leitung „*TD*“ zum Versand der Daten, „*RD*“ zum Empfang der Daten und die Masse „*GND*“ ausreichend.

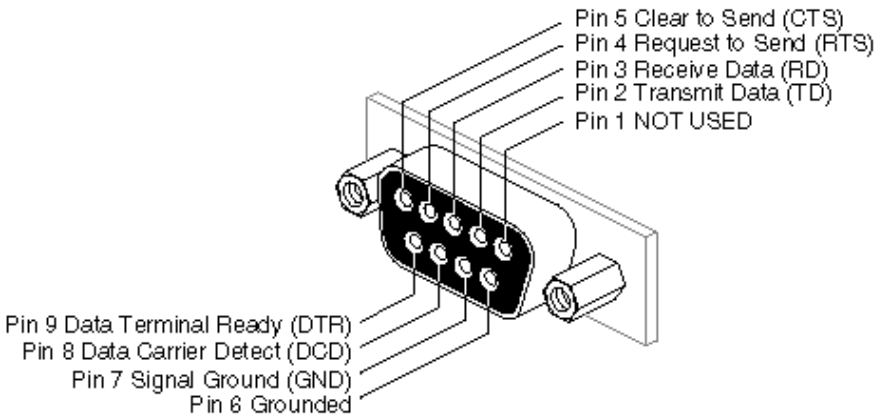

Abbildung 3.6: Belegung der EIA-232 Schnittstelle (Techbus , Zugriff: 20.04.2010)

Die Signalübertragung basiert auf einer negativen Logik. Im aktuellen 232-Standard (ANSI /EIA/TIA-232-F-1997) wird dafür ein Potential von ±15 V eingesetzt. Bei der verhältnismäßig großen Toleranz bilden die -3 V bis -15 V die logische eins und die +3 V bis +15 V die logische null. Dies ist in der nachfolgenden Tabelle 3.5 nochmal zusammengefasst:

Zustand	EIA-232 Potential
0	+3V bis +15V
1	-3V bis -15V

Tabelle 3.5: Logik der EIA-232 Schnittstelle

Ein Datenpaket kann bei der seriellen Schnittstelle auf unterschiedliche Weise definiert werden. Es können verschiedene *Baudraten*[19], Übertragungsbetriebe und unterschiedliche Definitionen der Bitreihenfloge, wird unter anderem für die Fehlererkennung benötig, eingestellt werden. Für ein erfolgreiches Zusammenspiel des Senders und des Empfängers müssen allerdings beide Seiten die gleichen Voreinstellungen haben. Nachfolgend werden die relevanten Einstellungen und Eigenschaften beschrieben:

Baudrate Baudrate wird als Bit pro Sekunde definiert. Bei der seriellen Schnittstelle können Baudraten in bestimmten Stufen von 2400 bis 115200 eingestellt werden.

Übertragungsart Hierbei muss zwischen einigen Übertragungsarten gewählt werden. Unteranderem ein Vollduplex-Betrieb, bei dem das Senden und Empfangen gleichzeitig möglich ist oder ein Halbduplex-Betrieb, bei dem das Senden und Empfangen nur

[19] Datenübertragungsrate der seriellen Schnittstelle

nacheinander durchgeführt wird. Ebenso kann zwischen einer *synchronen*[20] und asynchronen Übertragung unterschieden werden.

Startbit Jede Bitreihenfolge bei einer Übertragung beginnt mit einem Startbit, um den Beginn eines Datensatzes zu signalisieren.

Datenbits Die Datenbits werden hinter einem Startbit codiert und sind die eigentlichen Daten, die verschickt werden.

Paritätsbit Für die Fehlererkennung ist eine automatische Paritätsbitgenerierung möglich, die am Ende der gesendeten Bitreihenfolge angehängt wird. Das Paritätsbit symbolisiert eine bestimmte Anzahl von den übertragenden Bits. Beim Empfang wird das Paritätsbit ebenfalls untersucht und somit festgestellt, ob Daten verloren gegangen sind oder nicht.

Stoppbit Am Ende jeder Übertragungsfolge wird entweder ein Stoppbit oder für eine genauere Indentifinkation zwei Stoppbits angehängt.

Die serielle Schnittstelle des I/O Boards des Robotinos ist mit den folgenden Einstellungen vordefiniert.

Übertragung: asynchron
Betriebsart: vollduplex
Baudrate: 115200 $\frac{Bit}{s}$
Datenbits: 8
Paritätsbit: keins
Stoppbit: ein

3.1.5.3 Vereinfachtes Beispiel

Für eine bessere Veranschauung wird die Bitreihenfolge des ersten Bytes „R“ (binär: „0101 0010“) des Datenbyteblocks zur Steuerung des Robotinos (siehe Tabelle 3.3) in der Abbildung 3.7 unter Berücksichtigung der genannten Einstellungen visualisiert:

[20] Aufeinander abgestimmte Datenübertragung von Sender und Empfänger

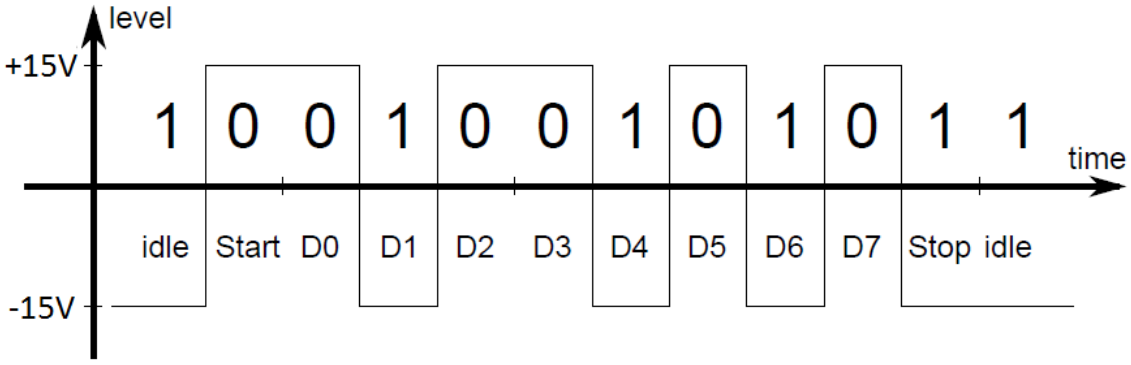

Abbildung 3.7: Bitreihenfolge des Buchstaben „R“ der seriellen Schnittstelle

3.1.6 Universal Serial Bus

Der Universal Serial Bus wurde erstmals für eine vielseitige und benutzerfreundliche Verwendung von externen Devices entwickelt. Das Hauptziel der Entwickler war die Realisierung eines einzigen physikalischen Anschlusses und die leichte Anwendbarkeit der anzuschließenden Geräte. Die Kommunikation zwischen einem Computer und einem Device läuft über eine Verbindung mit vier Strängen (siehe Abbildung 3.8).

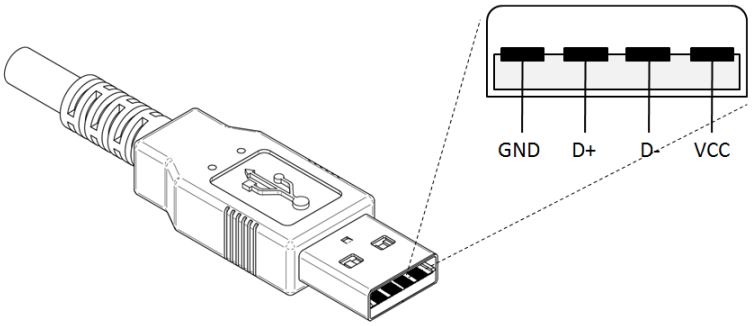

Abbildung 3.8: Pinbelegung eines USB Steckers (Wikipedia , Zugriff: 7.04.2010)

Die Masse „*GND*“ und die 5 V Spannungsversorgung „*VCC*“ speisen die angeschlossenen Geräte mit Energie. Dadurch bleibt in den meisten Fällen eine zusätzliche Spannungsversorgung erspart. Die Stränge „*D+*“ und „*D-*“ mit einem maximalen Spannungsunterschied von 4 V sorgen für eine differentielle Datenübertragung, bei der eine NRZI-Verschlüsselung benutzt wird. Bei dieser Art der Verschlüsselung kommt es auf einen Wechsel der Potentiale von „*D+*“ und „*D-*“ an. Ein Pegelwechsel symbolisiert dabei die binäre 0 und ein gleichbleibender Zustand die binäre 1. Anhand der Höhe der beiden Pegel wird zusätzlich zwischen einer „*Low-Speed*[21]“ und „*Full-Speed*“ Übertragung unterschieden. Derzeit weist der USB vier verschiedene Geschwindigkeiten auf, die sich im Laufe des USB Fortschritts gesteigert haben (siehe Tabelle 3.6).

[21] Hardwareabhängige Übertragungsgeschwindigkeit vom USB

Bezeichnung	USB	Geschwindigkeit
Low-Speed	USB 1.0/1.1	1,5 Mbit/s (187,5 KB/s)
Full-Speed	USB 1.0/1.1	12 Mbit/s (1,5 MB/s)
Heigh-Speed	USB 2.0	480 Mbit/s (60 MB/s)
Super-Speed	USB 3.0	5 Gbit/s (625 MB/s)

Tabelle 3.6: USB Geschwindigkeiten (Wikipedia , Zugriff: 7.04.2010)

Die Kommunikation über den USB zwischen dem Computer und dem angeschlossenem Gerät und damit auch die Codierung und Decodierung der Signale übernehmen *Controller*[22], die sich jeweils auf beiden Seiten befinden. Über die Device Seite ist nur schwer an Informationen zu gelangen, da die Unternehmen aufgrund von Geheimhaltung nur in seltenen Fällen *Spezifikationen*[23] ihrer Hardware veröffentlichen. Auf der Computerseite gibt es drei verschieden Controller, Enhanced Host Controller, der für USB 2.0 entwickelt worden sind, Universal Host Controller und Open Host Controller, die jeweils USB 1.0/1.1 übernehmen. USB 3.0 ist momentan noch in der Integrationsphase und ist nicht auf dem PC/104 des Robotinos vorhanden. Daher wird darauf nicht genauer eingegangen. Viele Computer haben einen Satz von diesen Host Controllern, um das ganze Spektrum abzudecken. Bei einem Anschluss eines USB Devices an einen Computer beginnt als erstes der schnellere Host Controller eine Kommunikation, bei dem die Eigenschaften des Devices abgefragt werden, unter anderem ob es sich um USB 2.0 oder USB 1.0/1.1 handelt. Falls der schnellere Host Controller nicht für dieses Device zuständig ist, da es nur USB 1.0/1.1 fähig ist, gibt er die Aufgabe an einen anderen Host Controller weiter. Anhand der erhaltenen Eigenschaften sucht der zuständige Host Controller nach passenden Treibern in der Datenbank des Computers und weist diese automatisch dem angeschlossenen Gerät zu. Diese Eigenschaft des USB wird auch „*Plug & Play*[24]“ genannt, wodurch eine manuelle Treiberinstallation unnötig wird. Die hin und her gesendeten Daten werden allgemein in vielen unterschiedlichen Paketen eingebunden, die abhängig von der Übertragungsart sind. Der USB agiert mit vier verschiedenen Übertragungsarten, die jeweils auf das angeschlossene Gerät abgestimmt werden. Darunter befindet sich ein „*Interrupt Transfer*“, geeignet für eine Tastatur oder Maus, ein „*Bulk Transfer*“, eingesetzt für externe Massenspeicher, ein „*Control Transfer*“ wird für das bereits beschriebene Verfahren der Eigenschaftenerkennung verwendet und der „*Isochrone Transfer*“, der zum Versand von Audiodaten oder Videodaten gebraucht wird. Der PC/104 des Robotinos besitzt auch zwei Host Controller, den Open Host Controller und den Enhanced Host Controller, die auf die Kommandobrücke herausgeführten USB Anschlüsse steuern. In der nachfolgenden Tabelle 3.7 werden im direkten Vergleich mit der EIA-232 Schnittstelle die Vor- und Nachteile des USB verdeutlicht:

	Geschwindigkeit	Komplexität	Verfügbarkeit
USB	480 Mbit/s (USB 2.0)	sehr komplex	in jedem Computer

[22] Elektronisches Gerät zur Steuerung von verschiedenen Vorgängen
[23] Technische Beschreibung eines Systems
[24] Hardwaregebrauch ohne zugehörige Konfigurationen

EIA-232	115200 Bit/s	gering	wird nicht mehr eingebaut

Tabelle 3.7: Vergleich EIA-232 und USB

Aus der Tabelle wird es deutlich, dass der USB eine viel schnellere Datenübertragung hat, dafür aber sehr viel komplexer in der Anwendung ist, als die serielle Schnittstelle. Der aktuelle Computerstandart sieht aber die Benutzung der EIA-232 nicht mehr vor und daher ist der Umstieg auf den USB oder ähnlich effiziente Schnittstellen nicht mehr umgänglich.

3.1.7 Wireless Local Area Network

Der PC/104 des Robotinos stellt einen LAN Anschluss bereit, der hauptsächlich für das Laden der Steuerprogramme vom *Client*[25] Computer zum Robotino und Übermittlung von Rückgabewerten vom Robotino zurück zum Client Computer eingesetzt wird. Der Client Computer ist in dem Fall ein Windows Rechner, auf dem die Steuerprogramme entwickelt werden und die Auswertung der zurückgegebenen Daten vorgenommen wird. Die Mobilität des Robotinos macht allerdings die Anwendung eines Kabels lästig. Mit Hilfe eines Access Points wird auf eine physikalische Verbindung verzichtet. Ein Access Point ist ein elektrisches Gerät, das eine Datenübertragung zwischen zwei Rechnern über Funk ermöglicht. Jedes Gerät bekommt eine Internet Protocol Adresse, durch die die Geräte in einem Netzwerk identifiziert und angesprochen werden können. Das Access Point dient dabei als ein Mittelstück, zu dem über eine IP Adresse eine Verbindung über Kabel sowie eine über Funk aufgebaut werden kann. Hierbei wird das Air Live 802.11G als ein solcher WLAN Adapter eingesetzt.

3.1.8 USB Webcam

Der Robotino verfügt eine Logitech E3500 Webcam, die an einem der beiden USB *Ports*[26] an der Kommandobrücke anschließbar ist. Diese Kamera kann in der Neigung und in der Drehung nach belieben verstellt werden. Der Einsatz der USB 2.0 Schnittstelle verspricht eine Rate von bis zu 30 Bildern pro Sekunde. Das Objektiv muss manuell scharf gestellt werden. Für eine bessere Bildaufnahme ist Softwaretechnisch eine Regelung der Lichtempfindlichkeit realisiert worden, sodass bei schwachen Lichtverhältnissen die Kamera mehr Licht durchlässt und bei starken wenig. An der Kamera ist noch ein Mikrofon vorhanden, mit dem zusätzlich akustische Signale aufgenommen werden können. Die Kamera schießt 24 Bit (True Color) Farbbilder mit einer maximalen Auflösung von 640 x 480 Pixel (15 Bilder/Sek). Leider werden keine detaillierten Spezifikationen der Webcams veröffentlicht, daher sind nur begrenzte Informationen vorhanden, die nochmal in der nachfolgenden Tabelle 3.8 zusammengefasst sind:

[25] Außenstehender Rechner, der zum Hauptrechner eine Datenverbindung herstellt
[26] Schnittstelle für externe Hardware

Bildsensor	CMOS-Farbsensor mit VGA-Auflösung
Farbtiefe	24 Bit (True Color)
Schnittstelle	USB 2.0
Auflösungen	160 x 120, 30 Bilder/Sek 176 x 144, 30 Bilder/Sek 320 x 240, 30 Bilder/Sek 352 x 288, 30 Bilder/Sek 640 x 480, 15 Bilder/Sek
Objektiv	manuell
Mikrofon	verfügbar

Tabelle 3.8: Technische Daten der Webcam (Logitech , Zugriff: 18.06.2010)

3.1.8.1 Bildeigenschaften

Ein aufgenommenes Bild hat eine maximale Auflösung von 640 x 480 Pixel, die von oben links nach unten rechts gezählt werden. Bei einem Farbbild, wie in diesem Fall, besteht jedes Pixel aus drei Bytes, die jeweils eine Farbintensität von Rot, Grün und Blau nachbilden. Angesichts der Bytegröße schwankt diese Intensität zwischen 0 bis 255, wobei 0 schwarz ist und 255 die jeweilige Grundfarbe. Aus der Mischung aller drei Grundfarben wird, am Beispiel eines *RGB-Farbraums*[27], jede mögliche Farbe zusammengestellt. Basierend auf diesem Hintergrund besteht das aufgenommene Bild aus drei übereinanderliegenden Ebenen mit 640 x 480 x 3 Bytes. Um Speicherplatz einzusparen, gibt es viele Möglichkeiten eine Codierung der Bytes vorzunehmen. Angesichts der Eigenschaften des menschlichen Auges wird darüber hinaus sogar auf bestimmte Informationen verzichtet. Die derzeit bekannteste Komprimierung ist Joint Photographic Experts Group (jpeg). Das einfachste unkomprimierte Bildformat ist Portable PixMap (ppm). Diese Bilddatei besteht aus einem *Header*[28], in dem ein Formatzeichen und die Bildgröße eingefügt sind und den eigentlichen, uncodierten Farbbytes. Das Formatzeichen im Header teilt dem zu lesenden Programm mit, um welche Art des Bildes es sich handelt, ob es beispielsweise schwarzweiß oder farbig ist. Zur Veranschaulichung des Speicherunterschieds zwischen einem komprimierten und einem unkomprimierten Bild wurde ein Aufnahme mit der Auflösung von 640 x 480 Pixel im ppm Format gemacht und abgespeichert. Das gleiche Bild wurde nachträglich erneut aufgerufen und im jpeg Format gespeichert.

[27] Farbdarstellung aller Farbvariationen durch Mischung der drei Grundfarben
[28] Kopfzeile einer Datei

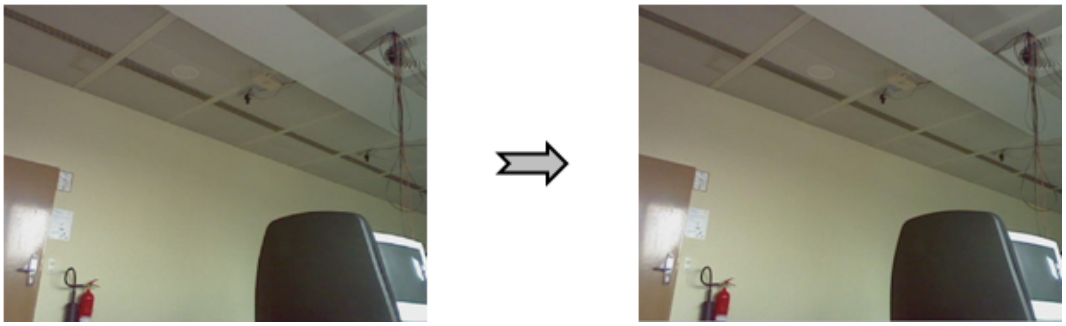

Abbildung 3.9: Komprimierung von ppm zu jpeg

In der Abbildung 3.9 sind die beiden Bildformate nochmal dargestellt. Auf den ersten Blick sind keine Unterschiede in den Bildqualitäten feststellbar. Wenn aber die Bildgrößen betrachtet wird, wurde das ppm Format von den 640 x 480 x 3 = 921600 Bytes beim Speichern in das jpeg Format auf 22500 Bytes komprimiert. Dies entspricht einer Speicherplatzeinsparung von ca. 98%. Für eine digitale Bildverarbeitung sind jpeg Formate leider nicht geeignet und daher für diese Zielsetzungen nur die uncodierten Bildformate verwendbar.

3.1.9 RobotinoView

Zur Ansteuerung des Robotinos wird von der Firma Festo Didactic eine individuelle Software namens „*RobotinoView*“ mitgeliefert. Diese Software wird auf einen Windows Rechner installiert und beinhaltet starre Steuerblöcke, die nur zusammengesetzt werden können, aber nicht programmierbar sind. Diese Blöcke greifen auf vordefinierte Funktionen zurück, die zusätzlich zu der „*RobotinoView*“ Software installiert werden müssen. Das Ganze funktioniert dann hauptsächlich über drei Ebenen hinweg. Auf der ersten Ebene „RobotinoView“ wird ein graphischer Aufbau der Steuerblöcke vorgenommen. Diese Blöcke greifen in der zweiten Ebene auf die vorinstallierten Funktionen zu, die über WLAN an den Robotino gesendet werden. In der letzten Ebene verarbeitet der PC/104 die ankommenden Steuerzeichen und setzt den Roboter in Bewegung. Wegen der unflexiblen Programmierung wird diese Software nur für Demonstrationszwecke eingesetzt und nicht nachfolgend verwendet.

3.2 Software

Für demonstrative Ansteuerung des Robotinos wurde bereits eine Software von Festo Didactic vorgestellt. Für die anstehende Aufgabenstellung ist diese Software jedoch nicht optimal. Aus diesem Grund werden andere Entwicklungsumgebungen verwendet, damit die Zielsetzungen dieser Arbeit erfüllt werden können.

3.2.1 Matlab

Bezugnehmend auf die weltweite Etablierung und den derzeitigen Einsatz an der HAW für Lösungen von Automatisierungsprozessen und Visualisierungen, bildet Matlab/Simulink in diesem Fall die Grundbasis der Ansteuerung des Robotinos. Diese Software wird von MathWorks regelmäßig aktualisiert und für Studenten kostenlos zur Verfügung gestellt. In dieser Diplomarbeit wird die Version R2009b mit den zugehörigen Toolboxen verwendet. Matlab besitzt über eine eigene, auf Matrizen basierte, Programmiersprache oder kann weitere Sprachen wie C und Fortran unterstützen. Eine große beigefügte Funktionsbibliothek bietet die Möglichkeit bestimmte Aufgaben und Visualisierungen zu vereinfachen. Beispielsweise sind viele Operationen der digitalen Bildverarbeitung nur mit einem einzigen Aufruf durchführbar.

3.2.1.1 Simulink

Zur Grundausstattung von Matlab gehört Simulink, welches eine graphische Entwicklungsumgebung bietet. In dieser Umgebung werden Daten kontinuierlich oder diskret über visualisierte Verknüpfungen weitergeleitet und verarbeitet. Die Verarbeitung der Daten wird mit Funktionsblöcken aus vorhandenen Bibliotheken vorgenommen. Zusätzlich bieten eigene Blöcke die Möglichkeit einen individuellen Programmcode einzufügen. Dieser Code kann entweder aus der Matlab eigenen Programmiersprache oder den zusätzlich unterstützen Sprachen bestehen. Mit diesen Eigenschaften sind Nachbildungen von realen Systemen realisierbar, sodass deren Verhalten simuliert untersucht werden kann.

3.2.2 xPC Target

Das xPC Target ist eine Toolbox aus Matlab/Simulink, die eine Echtzeitansteuerung einer seriellen Schnittstelle, wie der EIA-232 oder eine Datenübertragung über den LAN Anschluss bietet.

3.2.2.1 Funktionsweise

Der Aufbau funktioniert dabei so, dass auf einem *Hostrechner*[29] mit Matlab/Simulink die vorgefertigten I/O Funktionsblöcke für die Ansteuerung der jeweiligen Komponenten in die graphische Entwicklungsumgebung von Simulink eingefügt werden. Die Ein- und Ausgänge dieser Blöcke können zusätzlich durch eigen erstellten Funktionen Vor- und Nachbearbeitet werden. Die letztendlich erstelle Umgebung wird mit einem C/C++ Compiler, der auf dem gleichen Hostrechner vorhanden sein muss, zu einem Ausführbaren Code kompiliert und über eine LAN Verbindung auf den Target Rechner (xPC Target) geladen. Der Target Rechner läuft mit einem FreeDos Betriebssystem, auf dem ein Programm installiert ist, das

[29] Hauptrechner zu dem außenstehende Rechner eine Datenverbindung herstellen können

die kompilierten Daten empfängt und Ausführt. Im Anschluss an diesen Prozess wird eine Variable mit den entstandenen Rückgabewerten an den Host Rechner zurückgesendet und dort wieder mit Matlab/Simulink anhand von *Plots*[30] visualisiert und untersucht.

3.2.2.2 Anwendbarkeit

Nach diesem Verfahren wurde im Vorgängermodell eine Echtzeitregelung des Robotinos von Herrn Dipl.-Ing. Elard Köhler in seiner Diplomarbeit „*Entwicklung einer kamera-basierten Bahnführungs-Regelung für einen omnidirektionalen Roboter mit Matlab xPC Target*“ entwickelt. Angesichts der vielfältigen und komplexen Ansteuerungen der USB Devices, besitzt das xPC Target keine Anbindung einer Webcam an dieses Verfahren. Die Entwicklung eines hierzu benötigten Treibers erstellt sich, angesichts der fehlenden Spezifikationen der Geräte und der nicht vorhandenen Laborausrüstung, als schwer realisierbar.

3.2.3 Real-Time Workshop Embedded Coder

Der Real-Time Workshop Embedded Coder ist eine weitere Toolbox von Matlab/Simulink. Der ERT Coder konvertiert ein Simulinkmodell in ein von Matlab unabhängigen C-Code, der dann auf einem anderen Zielsystemen ausgeführt werden kann. Da die Kompilierungsregeln sich vom System zum System unterscheiden, werden „*Target Template Files*“ eingesetzt, in denen die Eigenschaften und die damit verbundenen Kompilierungsabläufe des jeweiligen Systems definiert werden. Ähnlich wie bei dem xPC Target können auch hier Segmente aus der Matlab Bibliothek eingesetzt werden. Der „*Matlab Embedded Coder*“ Block aus der Simulinkbibliothek bietet zusätzlich die Möglichkeit einen eigen geschriebenen Matlab Programmcode in diese Entwicklungsumgebung zu integrieren. Zusammengefasst kann ein Simulinkmodell in einen C-Code konvertiert werden, auf einen externen Zielsystem, wie ein Mikrokontroller oder einen PC/104 kopiert werden und dort kompiliert und ausgeführt werden.

3.2.3.1 Anwendbarkeit

Unter bestimmten Umständen bietet dieses Prinzip für die Zielsetzungen dieser Diplomarbeit eine weitere Aussicht die Aktoren und Sensoren des Robotinos anzusteuern. Es muss folglich ein Zielsystem gefunden werden, mit dem dieses Verfahren vereinbart werden kann und das die zugehörigen Punkte, wie die USB Unterstützung, erfüllt.

3.2.4 Linux

Linux ist ein öffentliches Betriebssystem, das im Vergleich zu Microsoft Windows ehe seltener in privaten Haushalten verwendet wird, aber dennoch lösungsorientiert in der Industrie

[30] Darstellung einer mathematischen Funktion

eingesetzt werden kann. Zu den wesentlichen Bestandteilen des Systems gehören:

Kernel: Der Kern des Betriebssystems, der die gesamte Verwaltung der Hard- und Software übernimmt. Unter anderem beinhaltet er die Treiber, die für die Steuerung der Schnittstellen zuständig sind.

Programme: Es gibt viele zusätzlich Programme, die für Linux entwickelt worden und im Internet frei verfügbar sind. Darunter befinden sich Programme zum Empfang und Visualisierung von Videodaten (Webcam), Ansteuerungen verschiedener Schnittstellen (USB, EIA-232) und Erstellung und Manipulation von Software (C-Compiler).

Shell: Die Shell ist eine Benutzerschnittstelle, über die Befehle ausgeführt und zugehörige Antworten dargestellt werden können. Nur der „*Super User*[31]“ hat das Recht alle Operationen durchführen zu können und kann anderen Benutzern Teilrechte vergeben. Zu den wesentlichen Punkten gehört die Navigation unter den Systempfaden, die Installation zusätzlicher Programme und das Kompilieren und Ausführen von C-Codes.

X: X ist eine grafische Systemumgebung zur Navigation der Systempfade, ähnlich des Desktops unter Windows.

3.2.4.1 Schalenmodell

Die Linuxarchitektur kann anhand eines Schalenmodells aus der nachfolgenden Abbildung 3.10 beschrieben werden:

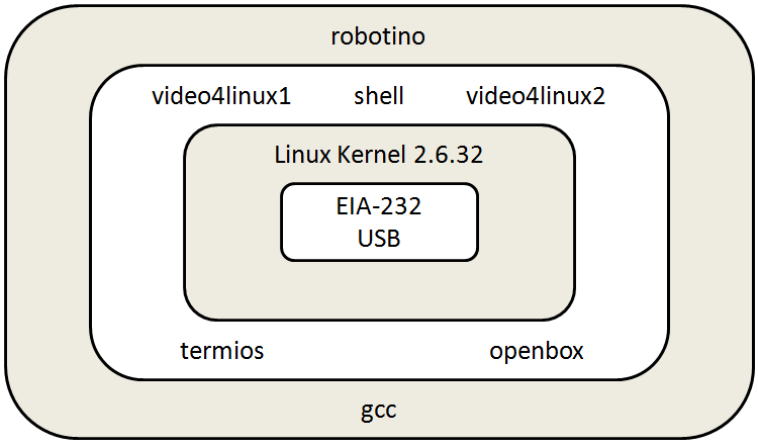

Abbildung 3.10: Linux als Schalenmodell

Im Kern des Modells befindet sich die Hardware, die mittels der überliegenden Schichten gesteuert wird. Als Beispiel wird hier der USB und die serielle Schnittstelle aufgeführt. Die

[31] Hauptbenutzer eines Linux Betriebssystems

Hardware ist bei diesem System von einem Linux Kernel umschlossen, der mit grundlegenden Hardwaretreibern die wesentlichen Systemkomponenten Verwaltet. Die Kernelfunktionen sind geschützt und können von den Benutzern des Linux Betriebssystems nur in speziellen Fällen manipuliert werden. Aufgrund dieser Eigenschaft werden Linux Betriebssysteme als stabil bezeichnet und dementsprechend vermehrt für *Embedded Systeme*[32] verwendet. Der Kernel greift auf die überliegende Schicht zu, um weitere Funktionen für die Verwaltung der Hardware einzubinden. Diese Schicht kann unter anderem Bibliotheken für Webcams und die EIA-232 Schnittstellen, Desktopumgebungen wie openbox oder Terminals zur Eingabe von Linuxbefehlen, enthalten. In der obersten Schicht befinden sich die Applikationen, die von den Benutzern installiert und manipuliert werden können. Hierbei können eigen erzeugte Programme eingebunden werden, die Schichtübergreifend Daten aus der Hardware aufnehmen und wieder zurückschicken können.

3.2.4.2 Linuxbefehle

Ein Linux Befehl wird in einer Shell eingegeben und besteht aus einem Programmnamen mit Optionen und Argumenten. Das Programm befindet sich in einem Verzeichnis, dessen Pfad in dem Namen mit angegeben werden muss. Die Optionen werden hinter dem Programmnamen hinzugeschrieben, beginnend mit einem Bindestrich. Mit ihnen können Einstellungen der jeweiligen Programme vorgenommen werden. Im Anschluss der Befehlseingabe kommen die Argumente, die für Ein- und Ausgaben benutzt werden. In der nachfolgenden Tabelle 3.9 werden einige der wichtigsten Befehle erläutert:

Allgemein		
[Befehl] &	Man kann die Konsole weiter benutzen	firefox &
[Befehl] && [Befehl]	Befehle werden nacheinander ausgeführt	firefox && clear
man [Programm]	Hilfe zu einem Programm	man firefox
strg + C	Aktuellen Vorgang in der Shell abbrechen	strg + C
Verzeichnisse, Dateien		
cd [Verzeichnis]	In das Verzeichnis wechseln	cd /tmp
cd ..	Ein Verzeichnis höher wechseln	cd ../td>
cd /	In das Wurzelverzeichnis wechseln	cd /
cd -	Wechsel in das vorherige Verzeichnis	cd -
cp [Datei][Verz]	Kopiert Datei in Verzeichnis	cd 123.txt /tmp
mv	Verschiebt eine Datei	mv 123.txt /tmp
mv [Datei1][Datei2]	Benennt Datei1 in Datei2 um	mv 123.txt 456.txt
rm	Löscht eine Datei	rm 123.txt
rm -rf	Alles unterhalb des Verz. löschen	rm -rf /tmp/
mkdir	Erzeugt ein Verzeichnis	mkdir /home/test
rmdir	Löscht ein Verzeichnis	rmdir /home/test
ls	Zeigt Ordnerinhalt an	ls /home/test

[32] Eingebundene Rechensysteme für Automatisierungsprozesse

ls -l	Ausführiche Auflistung des Ordnerinhalts	ls -l /home/test
ls -la	Dateien des Verz. ausführlich anzeigen	ls -la /home/test
alias ls='ls –color'	Stellt farbige Ansicht ein	alias ls='ls –color'
pwd	Zeigt das aktuelle Verzeichnis an	pwd
cat [Datei]	Zeigt Inhalt einer Datei	cat 123.txt
more [Datei]	Zeigt Inhalt einer Datei seitenweise an	more 123.txt
touch [Datei]	Erzeugt leere Datei	touch 123.txt
whereis [Prog]	Sucht nach Programm	whereis firefox
find . | grep [Datei]	Sucht eine Datei im Verzeichnis	find . | grep 123.txt
grep [Key][Datei]	Sucht nach Begriff in einer Datei	grep Haus 123.txt
locate [Datei]	Sucht nach Datei in der Datenbank	locate 123.txt
updatedb	Aktualisiert die Datenbank	updatedb
which	Zeigt wo sich ein Progr. befindet	which firefox
System		
arch	Prozessorfamilie	arch
cat /proc/filesystems	Unterstütze Dateisysteme	cat /proc/filesystems
cat /proc/cpuinfo	Infos zur CPU	cat /proc/cpuinfo
cat /proc/pci	Infos zu den PCI-Karten	cat /proc/pci
dmesg | grep hd	Infos über alle Laufwerke	dmesg | grep hd
date	Datum und Zeit	date
dmesg	Kernelloger: Zeigt Kernelaktivitäten	dmesg
free	Zeigt Ausnutzung des Arbeitsspeichers	free
glxgears	Kleiner Grafiktest	glxgears
kill [PID]	Schießt Prozess mit bestimmter ID ab	kill 1067
killall [Progr]	Schießt Prozess mit Prozessname ab	killall firefox
lspci	Infos über PCI-Komponenten	lspci
shutdown -h now	Fährt den Rechner herunter	shutdown -h now
shutdown -r now	Startet den Rechner neu	shutdown -r now
top	Zeigt CPU-Auslastung an	top
uptime	Zeigt Betriebszeit des PC's an	uptime
X-version	Zeigt Version von Xfree an	X-version
Benutzer		
id	Zeigt Benutzername und Gruppe an	id
whoami	Zeigt aktuell angemeldeten Benutzer an	whoami
who	Zeigt alle eingeloggten Benutzer	who
groupadd [Gruppe]	Erzeugt eine neue Gruppe	groupadd admins
groupdel [Gruppe]	Löscht eine Gruppe	groupdel admins
useradd -m [User]	Erzeugt Benutzer und Homeverzeichnis	useradd -m admin
userdel -r [User]	Löscht Benutzer und Homeverzeichnis	userdel -r admin
passwd [User]	Ändert Passwort des Benutzers	passwd admin
su	Als Super-User an der Konsole arbeiten	su
su [User]	Als Benutzer an der Konsole arbeiten	su admin

Netzwerk		
ifconfig	Zeigt Netzwerkinfos an	ifconfig
iwconfig	Zeigt Infos zum WLAN an	iwconfig
ping [IP]	Test der Verbindung zu einem Rechner	ping 192.168.0.12
Kernel und Module		
lsmod	Zeigt geladene Module an	lsmod
make menuconfig	Einrichten des Kernels	make menuconfig
modprobe [Modul]	Lädt ein Modul	modprobe printer
uname -a	Zeigt Kernelversion an	uname -a
Sonstiges		
ps aux	Zeigt alle laufenden Prozesse an	ps aux
rc-update show	Zeigt Dienste beim Start an	rc-update show

Tabelle 3.9: Zusammenfassung der wesentlichen Linuxbefehlen (PC-Erfahrungen , Zugriff: 14.04.2010)

3.2.4.3 Weitere Eigenschaften

Die Installation von Programmen unter Microsoft Windows läuft immer über ausführbare „*.exe*[33]“ Dateien ab. Während dessen werden Dateien, unter denen sich wiederrum andere ausführbare Dateien und Bibliotheken befinden, in vorgesehene Verzeichnisse des Windows Rechners kopiert. Die meisten Programme sind nicht öffentlich und werden von den Herstellern geschützt. Die kopierten Verzeichnisse werden somit codiert, vor allem die Funktionsbibliotheken, sodass Benutzer keine Möglichkeit haben sich deren Struktur anzuschauen. Programminstallationen unter Linux funktionieren nach einem etwas andren Prinzip. Da Linux ein öffentliches Betriebssystem ist, werden die Funktionsstrukturen der Applikationen nicht versteckt. Sie sind alle in der Programmiersprache C geschrieben und frei einsehbar. Ein dazugehöriges „*makefile*“ ist das Kernstück jeder Installation. Diese Datei enthält die Regeln und Pfade, wie die Kompilierung und Verlinkung der Programmcodes ablaufen soll. Durch einen Aufruf dieser Datei mit einem Linux C-Compilers, wird der Funktionsinhalt der Programme kompiliert und die Programme installiert.

3.2.4.4 Linux im Vergleich

Die signifikanten Eigenschaften bieten bei einem Linuxsystem, im Vergleich zu den Windows Rechnern, deutliche Vorteile für Entwickler. Durch die öffentliche Einsicht in die Programmstrukturen werden Fehler viel schneller entdeckt und können von jedem beseitigt werden. Aus diesem Grund gilt Linux als ein viel stabileres Betriebssystem, als seine Konkurrenten. Eine frei zugängliche Manipulation hat im Laufe der Zeit eine beträchtliche Variation dieser Betriebssysteme hervorgebracht, die jedoch alle auf demselben Kernel

[33] Ausführbare Dateien unter Windows

aufbauen. Dieser Kernel wird regelmäßig an den derzeitigen Computerstandart angepasst, sodass die komplette Bandbreite der aktuellen Hardware unterstützt wird. Eine große Benutzergemeinschaft fördert diese Entwicklung und hilft bei vielen Problemlösungen.

Im Bezug auf die Aufgabenstellung dieser Diplomarbeit, unterstützen Linux und Windows die geforderten Schnittstellen und verfügen über Funktionsbibliotheken, mit denen Webcams angesteuert werden können. Im direktem Vergleich in der nachfolgenden Tabelle 3.10 von Windows 7 und dem meist genutzten Linux Betriebssystems Ubuntu wird veranschaulicht, dass Windows als Betriebssystem für den Robotino nicht geeignet ist:

Anforderungen	**Windows 7**	**Ubuntu Lynx**
Minimale Prozessorgeschwindigkeit	1 Ghz	1 Ghz
Minimaler Arbeitsspeicher	1 GB	1 GB
Benötigter Festplattenspeicher	16 GB	15 GB
USB 1.0/1.1 und USB 2.0 Unterstützung	ja	ja
Webcam Unterstützung	ja	ja
EIA-232 Unterstützung	ja	ja

Tabelle 3.10: Vergleich zwischen Windows 7 und Ubuntu (Ubuntu , Zugriff: 17.03.2010)

Die aufgelisteten Daten aus der obigen Tabelle 3.10 zeigen, dass die Voraussetzungen für ein Windows 7 Betriebssystem, hinsichtlich der PC/104 Werte, nicht erfüllt werden. Die Angaben für Ubuntu wurden für die komplette Systemarchitektur aufgeführt. Mit der Eigenschaft einer individuellen Linuxmodellierung kann der Festplattenspeicher drastisch und die minimalen Anforderungen für die Prozessorgeschwindigkeit sowie den Arbeitsspeicher teilweise gesengt werden. Es gibt noch kleinere Linuxsysteme wie das „*Damn Small Linux*“, die nur 100 MB Speicherplatz benötigen, jedoch fehlen diesen Systemen wichtige Bestandteile für die anstehende Problematik. Deren Installation würde den Speicherbedarf an die umfassenden Linux Distributionen angleichen. Ein großer Vorteil der Ubunu *Distribution*[34] bietet die einfache Handhabung der Programminstallation, die nur mit einem Befehl in der Shell ausgeführt werden kann. Dazu kommt noch eine ausgiebige Funktionsbibliothek und eine umfangreiche Benutzergemeinschaft, die über Internetforen eine gute Hilfestellung geben kann. Anläßlich dieser Vorteile bildet in dieser Diplomarbeit die minimale Ubuntu Lynx Distribution die Plattform für die Steuerprogramme des Robotinos.

3.2.5 Putty

Putty ist ein Programm für Microsoft Windows, mit dem eine Verbindung zu einem anderen Rechner mit einer Shell hergestellt werden kann. Der Verbindungsaufbau kann durch bestehende Methoden gesichert werden und infolgedessen einen illegalen Verbindungsversuch

[34] Unterschiedliche Linuxversionen, die auf dem gleichen Kernel aufgebaut werden

verbieten. Die physikalische Verbindung und die Kommunikation läuft über einen LAN beziehungsweise WLAN Anschluss. Die Adressierung wird dabei über die IP des Shellrechners abgewickelt. Beim Aufruf des Programms und nach einem erfolgreichen Verbindungsaufbau wird dem Benutzer ein *Terminal*[35] breitgestellt, in dem direkt die Befehle eingegeben werden können. Dieses Programm kann zur Fernsteuerung eines Linux Betriebssystem, in diesem Fall den Robotino, von einem Microsoft Windows Rechner benutzt werden.

3.2.6 WinSCP

WinSCP ist ein Windows Programm mit einer grafischen Oberfläche, das einen gesicherten Datentransfer zwischen mehreren Rechnern ermöglicht. Die Kommunikation wird über eine IP Adresse des Zielrechners initialisiert und läuft auf Grund dessen über einen LAN beziehungsweise WLAN Kanal. Bei einer erfolgreichen Verbindung wird dem Benutzer eine grafische Oberfläche dargeboten, die die Systemordner des Windows- und des Zielrechners anzeigt. Über diese Oberfläche können die Dateien durch „*Drag & Drop*[36]“ von einem System zum anderen geschoben werden. Mit diesem Programm werden somit Daten von einem Windows Rechner auf den Robotino, der unter Linux läuft, kopiert. Zusammen mit Putty kann so eine komplette Fernsteuerung eines Linux Betriebssystems und damit auch des Robotinos realisiert werden.

[35] Visuelles Fenster für die Eingabe von Befehlen
[36] Deteiverschiebung mit der Computermaus

4 Systemaufbau

Für die Realisierung der Steuerung und Regelung des Robotinos muss erst einmal das gesamte System mit den zugehörigen Komponenten aufgebaut und konfiguriert werden. Zu diesem Zweck wird ein passendes Betriebssystem auf den PC/104 des Robotinos installiert und an die Eigenheiten des Robotinos angepasst. Die Ansteuerung der gesamten Schnittstellen und der Hardware, unter andrem die serielle Schnittstelle und die USB Webcam, muss ebenfalls eingerichtet werden. Um das Beispiel der Vorgängermodelle aufgreifen zu können, wird eine Verknüpfung zwischen Matlab/Simulink und dem neuen Betriebssystem entwickelt. Die Benutzung des standardisierten Steuer- und Simulationstools bleibt dabei weiterhin bestehen. In diesem Kapitel werden die dafür benötigten Schritte durchgeführt und die zugehörigen Bestandteile beschrieben.

4.1 Installation von Ubuntu Lucid Lynx

Ubuntu Lucid Lynx bietet das ideale Betriebssystem zur Steuerung des Robotinos und seiner kompletten Hardware. Aus den vorherigen Kapiteln geht hervor, dass der PC/104 die minimalen Anforderungen dieses Betriebssystems noch nicht erfüllt. Die Anforderungen wurden jedoch für das komplette Betriebssystem angegeben, das heißt eine Installation mit der visualisierten Desktopumgebung und den häufig benutzten Programmen wie Firefox und MediaPlayer. Die Eigenschaft der individuellen Modularität von Linux bietet noch einen anderen Weg Ubuntu Lucid Lynx im minimalen Zustand zu installieren. Dabei wird der Kernel nur mit der Shell und der Funktionsbibliothek des Betriebssystems raufgespielt. Bei Bedarf kann die fehlende Software wieder nachgerüstet werden. Auf diese Weise werden die Anforderungen gesenkt und die Leistung des Systems verbessert. Im nachfolgendem Teil des Kapitels wird Schritt für Schritt gezeigt, wie die minimale Installation Ubuntu Lucid Lynx durchgeführt und was dafür benötigt wird.

4.1.1 Systemvoraussetzungen

Für die minimale Ubuntu Lucid Lynx Installation werden folgende Systemanforderungen benötigt:

- 300 MHz x86 Prozessor
- 256 MB RAM Systemspeicher
- 2 GB Festplattenspeicherplatz
- Grafikkarte mit einer Auflösung von 640 x 480

4.1.2 Vorbereitung

Für die Installation des minimalen Ubuntu Lucid Lynx Betriebssystems auf den Robotino werden folgende Hardwarekomponenten vorausgesetzt:

- 2 GB (empfohlen 4 GB) CF Speicherkarte
- CF Speicherkarten Lesegerät
- 700 MB CD Rohling
- Ubuntu Lucid Lynx Alternate-CD Image
- Offene Interneverbindung über LAN

Zuerst muss das *Ubuntu Lucid Lynx Alternate-CD Image*[37] aus dem Internet heruntergeladen werden. Die Quelle dafür befindet sich auf der folgenden Seite:

Download: http://wiki.ubuntuusers.de/Downloads/Lucid_Lynx

Alternativ können auch ältere Versionen von Ubuntu verwendet werden. Die heruntergeladene Datei ist ein ISO Format, welches auf eine 700 MB CD Rohling gebrannt werden muss. Im Anschluss darauf wird die CF Speicherkarte in das dafür vorgesehene Lesegerät reingesteckt und das ganze über ein USB Kabel mit einem Windows Rechner verbunden. Eine offene Internetverbindung wird benötigt, die am besten über ein LAN Kabel mit einem DSL Router hergestellt werden kann. Während der Installation werden darüber zusätzliche Pakete heruntergeladen. Nun wird die gebrannte CD mit dem Linux Betriebssystem in das Laufwerk eingelegt. Der Autostart der CD wird abgebrochen, da es für den weiteren Installationsablauf nicht von Bedeutung ist. Jetzt wird der Windows Rechner neu gestartet und mit der „*del*“ Taste, während des Bootvorgangs, das BIOS des PC's geöffnet. Das öffnen des BIOS ist vom *Mainboardhersteller*[38] abhängig und kann daher bei diesem Prozess Unterschiede aufweisen. Zum Schluss wird im BIOS das primäre Booten von der CD eingestellt und die Einstellungen gespeichert.

Diese Einstellungen bewirken einen Start des Installationsmenüs von Ubuntu Lucid Lynx von der eingelegten CD beim Bootvorgang des Computers. Damit ist die Installation des Betriebssystems auf die CF Speicherkarte vorbereitet worden und kann nun begonnen werden.

Hinweis: Bei der Installation gehen alle Daten auf der CF Speicherkarte verloren, daher wird empfohlen die vorhandenen Daten auf einem andren Laufwerk zu sichern. Eine fehlgeleitete Installation kann auf der Hauptfestplatte des Rechners das komplette Windows Betriebssystem löschen. Diesbetreffend muss beim weiteren Fortgang

[37] Alternative Installationsdatei für Ubuntu
[38] Hauptplatine eines Computers

sichergestellt werden, dass die Installation auf der CF Speicherkarte durchgeführt wird.

4.1.3 Installationsverlauf

Nach den Vorbereitungen wird der Windows Rechner neu gestartet und das Ubuntu Installationsmenü von der eingelegten CD gestartet (siehe Abbildung 4.1):

Abbildung 4.1: Im ertsen Aufruf des Installationsmenüs kann zwischen verschiedenen Sprachen gewählt werden, in diesem Fall „*Deutsch*“.

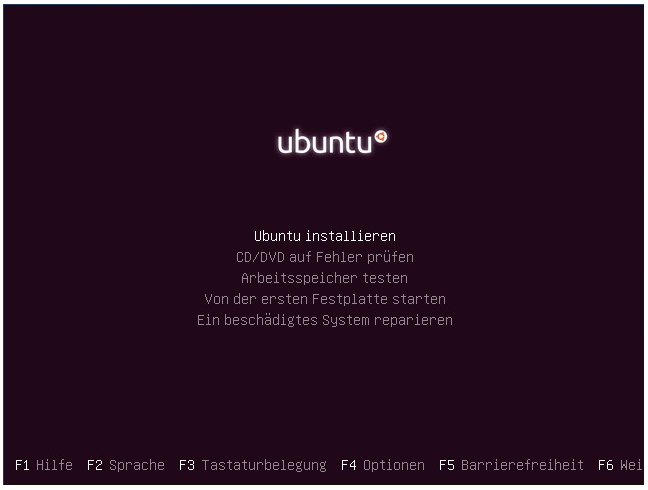

Abbildung 4.2: Für die minimale Installation müssen einige erweiterte Optionen eingestellt werden. Diese werden nach der Auswahl der Sprache im unteren Bildschirmrand eingeblendet.

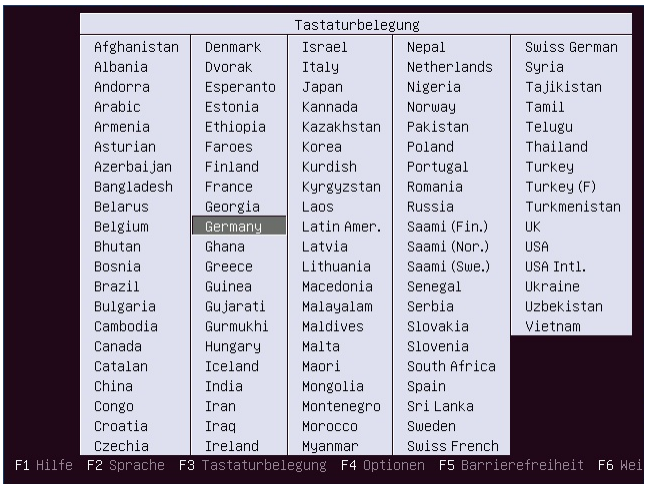

Abbildung 4.3: Mit der Taste „*F3*“ gelangt man zur Tastaturbelegung und wählt dort „*Germany*“ aus.

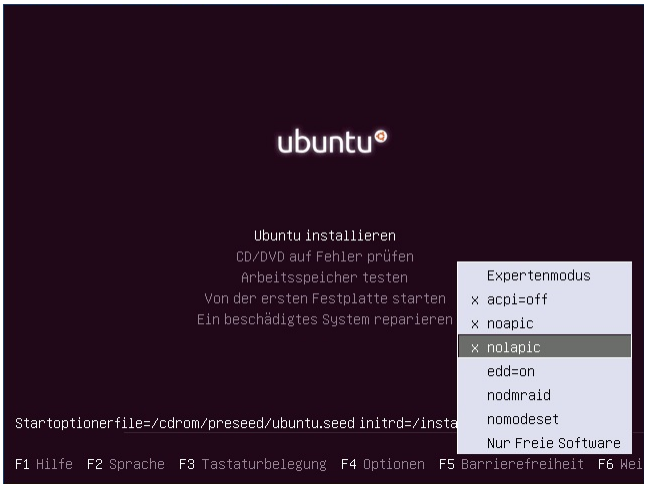

Abbildung 4.4: Mit der Taste „*F6*“ werden weitere Optionen geöffnet. Hier wird mit Hilfe der Leertaste „*acpi=off*“, „*noapic*“ und „*nolapic*“ gesetzt. Diese Optionen sind für den Robotino nicht relevant und werden deswegen ausgeschaltet.

Abbildung 4.5: Zum Schluss wird mit der Taste „*F4*“ „*Ein Befehlszeilensystem installieren*“ gesetzt und danach „*Ubuntu installieren*“ ausgeführt.

Abbildung 4.6: Bei einer erfolgreichen Verbindung zum Internet wird ein Rechnername ausgesucht. Dieser Name hat für den Robotino keine Bedeutung und daher wird der vorgeschlagene Eintrag „*Ubuntu*“ übernommen.

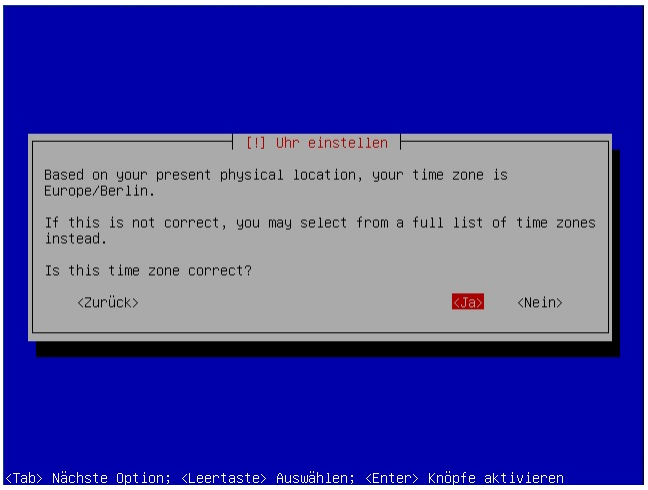

Abbildung 4.7: Beim weiteren Installationsverlauf wird nach der Zeitzone gefragt. Durch die Wahl der Sprache ist die europäsche Zeit bereits eingesetzt.

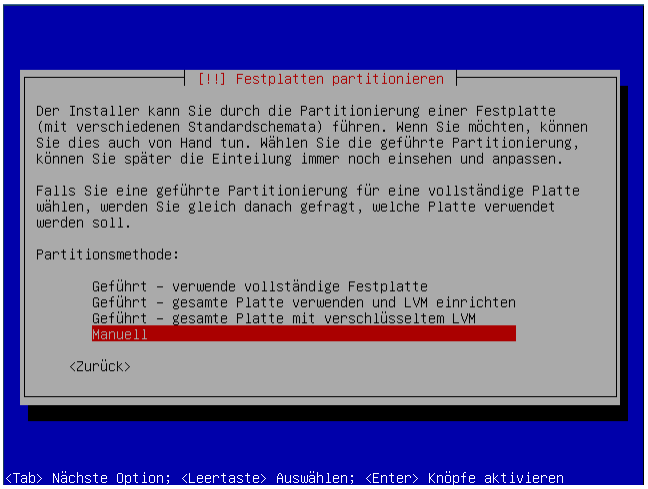

Abbildung 4.8: Da das System auf die CF Speicherkarte installiert werden soll, muss die manuelle Partionierung vorgenommen werden.

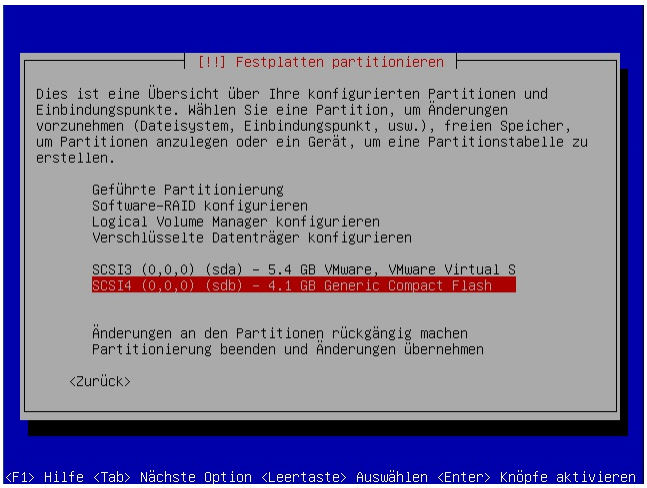

Abbildung 4.9: Hier werden alle verfügbaren Festplatten des Rechners dargestellt. In diesem Punkt ist darauf zu achten, dass die CF Speicherkarte gewählt wird. Eine falsche Auswahl könnte die Festplatte des Windows Rechners löschen.

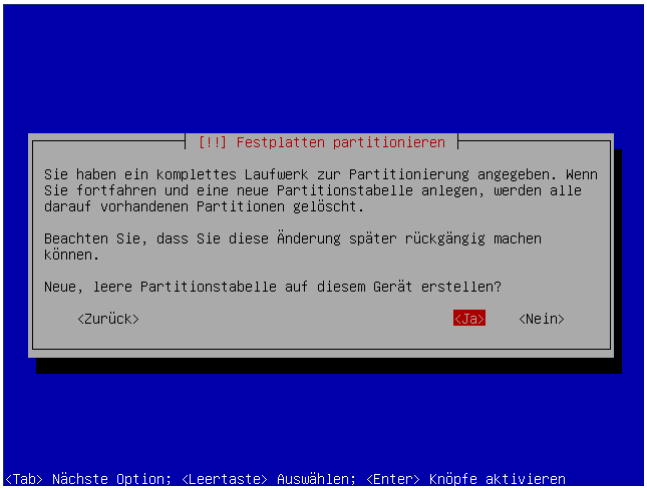

Abbildung 4.10: Nach der Auswahl der CF Speicherkarte wird die leere Partitionierung bestätigt und erstellt.

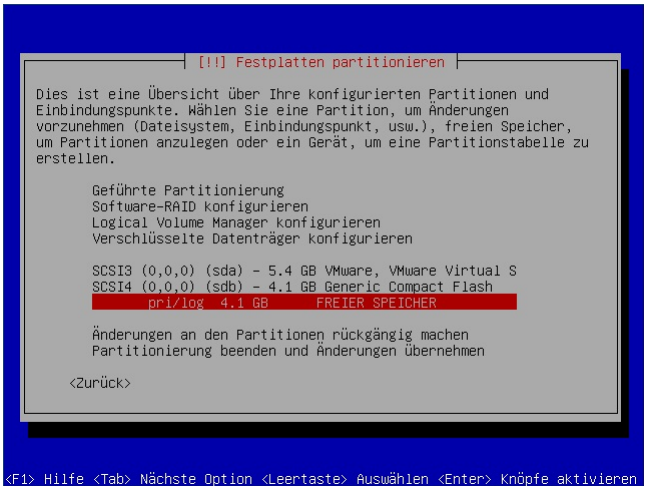

Abbildung 4.11: Diesmal ist eine leere Partition auf der CF Speicherkarte sichtbar, die jetzt ausgewählt wird.

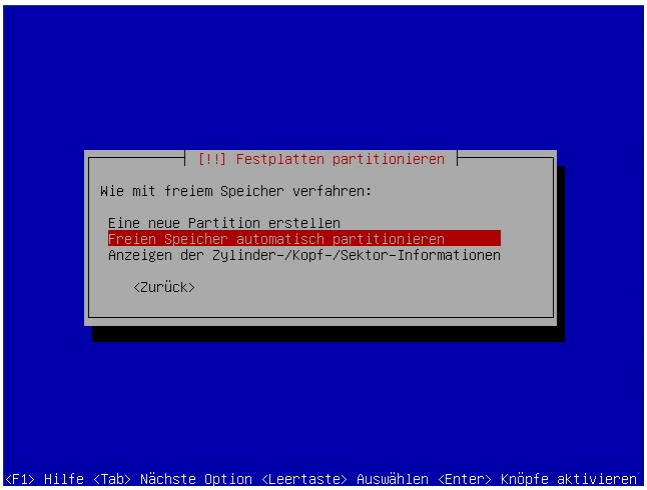

Abbildung 4.12: Im Folgenden Schritt wird für die Installation des Betriebssystems der freie Speicherplatz der CF Speicherkarte automatisch generiert.

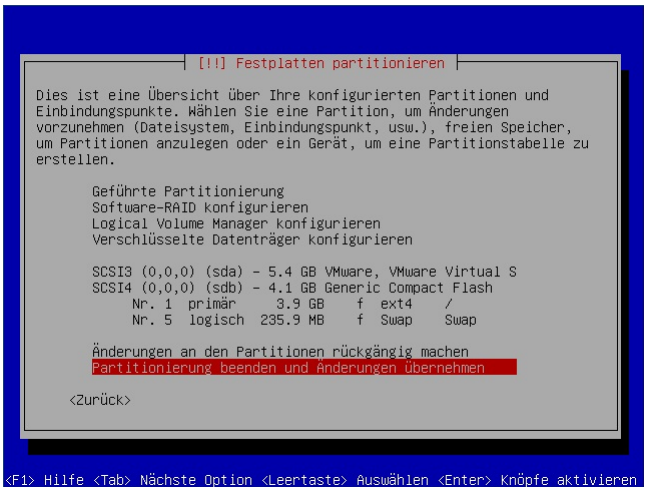

Abbildung 4.13: Nun wird die Änderung auf die CF Speicherkarte geschrieben und die Installation fortgeführt.

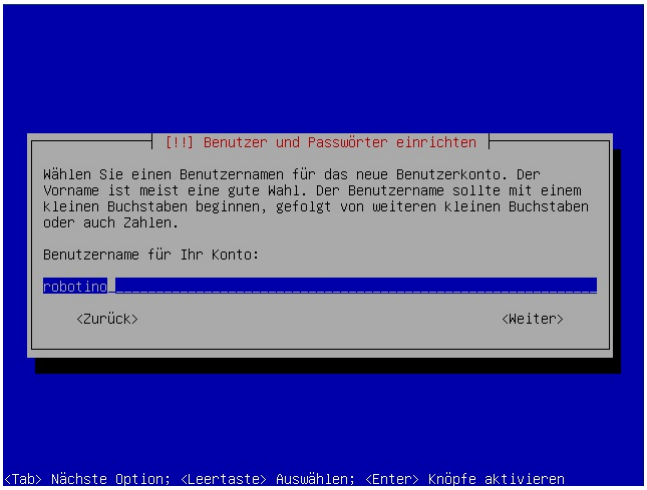

Abbildung 4.14: Hier wird der Benutzername „*robotino*“ gewählt. Bei der Fertigstellung der Installation wird für diesen Benutzer ein Verzeichnis erstellt, das später verwendet wird.

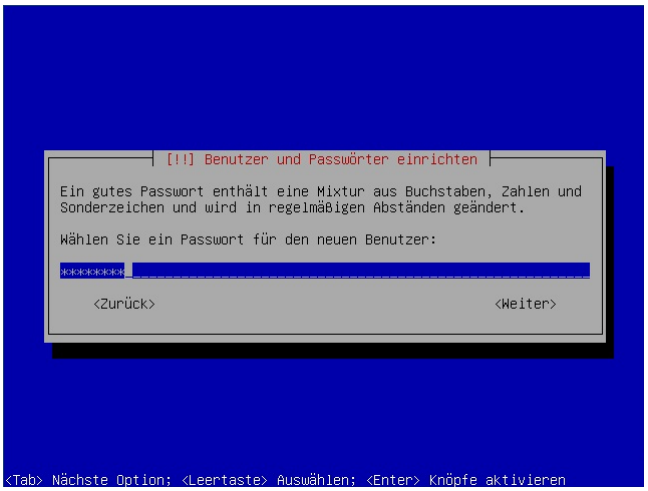

Abbildung 4.15: Zu dem Benutzernamen wird auch das Passwort „*robotino*“ ausgesucht und im darauf folgendem Bild nochmal bestätigt.

Abbildung 4.16: In den letzten Schritten wird auf eine Verschlüsselung und die Proxy-Daten verzichtet und das Schreiben des GRUB-Bootloaders in den Master Boot Record akzeptiert.

Damit ist die minimale Installation von Ubuntu Lucid Lynx auf die CF Speicherkarte abgeschlossen und kann nun für den Einsatz auf dem Robotino konfiguriert werden.

4.1.4 Konfiguration

Nach der Installation des Ubuntu Betriebssystems werden noch einige Konfigurationen durchgeführt, die das spätere Arbeiten mit dem Robotino erleichtern werden. Unter anderem werden Vorgänge wie der Systemlogin automatisiert und zusätzliche Programme für eine bequemere Bedienung nachinstalliert. Darüber hinaus werden Einstellungen vorgenommen, die wegen der besonderen Eigenschaften des Robotinos benötigt werden.

4.1.4.1 BIOS des Robotinos

Das Basic Input Output System ist ein Programm, das sich auf einem nichtflüchtigen

Speicher eines Mainboards befindet. Das BIOS übernimmt die Verwaltung der *Peripheriegeräte*[39], die an diesem Board angeschlossen sind. Die BIOS Systemuhr, die über eine Batterie betrieben wird, spielt eine entscheidende Rolle bei dem Uhrwerk der kompletten Hardware und Software. Zu beachten ist, dass der PC/104 keine Batterie besitzt und angesichts dessen wird die Systemzeit des Robotinos immer zurückgesetzt, sobald die Energiezufuhr unterbrochen wird. Das BIOS des Robotinos wird durch Drücken der „*del*“ Taste während des Bootvorgangs geöffnet. In dessen Einstellungen können manuell einige Hardwarekonfigurationen vorgenommen. Darunter ist der Bootvorgang von der CF Speicherkarte einzustellen, damit der Robotino mit dem neu installierten Ubuntu Betriebssystem starten kann. Weitere Optionen sind nicht relevant und bleiben daher unverändert.

4.1.4.2 Zusätzliche Programme

Die vorherigen Einstellungen im BIOS ermöglichen nun den Start des Linux Betriebssystems. Der *GRUB Bootloader*[40], der während der Installation (siehe Abbildung 4.16) in den *Master Record*[41] geschrieben worden ist, wird beim Start geöffnet und bietet eine zehn sekundige Auswahl zwischen mehreren Bootoptionen. Die Wahl des standard Bootvorgangs startet letztendlich das Linux Betriebssystem. Aufgrund der minimalen Installation von Ubuntu, steht nach dem Start nur die Shell zur Verfügung, bei der als erstes der Benutzername und das zugehörige Passwort abgefragt werden. Für diese werden jeweils „*robotino*“ verwendet, mit denen man sich im System als *root*[42] Benutzer anmeldet. Jetzt kann mit der Installation zusätzlicher Programme begonnen werden. Ubuntu bietet dabei eine sehr bequeme Methode dies zu bewerkstelligen. Über eine Internetverbindung und mit einem einzigen Befehl in der Shell

```
sudo apt-get install [Programmname]
```

werden die Programme mit den zugehörigen Abhängigkeiten heruntergeladen und automatisch installiert. Die Anweisungen mit „*sudo*“ am Anfang der Eingabe geben den eingegebenen Befehlen die Rechte des Hauptbenutzers. Deshalb wird beim Aufruf dieses Befehls nochmals nach dem Benutzernamen und dem Passwort gefragt. Denn der Hauptbenutzer ist der einzige, dem es gestattet ist neue Programme zu installieren.

Für eine einfachere Bedienung des Betriebssystems auf dem Robotino wird die zusätzliche Installation folgender Pakete empfohlen:

- „*openbox*“ ist ein simpler grafischer *Fenstermanager*[43] für Linux, der weinig Ressourcen benötigt und mit dem Behfehl

  ```
  sudo apt-get install openbox
  ```

[39] An der Hauptplatine angeschlossene Geräte eines Computers
[40] Programm zum Start des Linux Betriebssystems
[41] Der erste Datenblock eines Speichermediums
[42] Hauptbenutzer eines Linux Betriebssystems
[43] Bildschirmumgebung für die visuelle Verwaltung von Programmen

auf den Robotino installiert wird. Damit wird der eigentliche beigefügte Fenstermanager „*Gnome*“, der bei der Vollversion von Ubuntu beigefügt ist, ersetzt. Gnome ist für eine hochauflösende Darstellung vorgesehen und würde die Anforderungen des Betriebssystems für den PC/104 sprengen.

- Openbox enthält keine zusätzliche Software für die Bedienung des Betriebssystems. Um die Shell weiterhin nutzen zu können wird „*xterm*“ benötigt, der die Eingabe von Befehlen auf der grafischen Oberfläche bereitstellt. Mit dem Aufruf

  ```
  sudo apt-get install xterm
  ```

  wird ein Terminal heruntergeladen und installiert, der dies einrichtet. Ähnlich der Windows Eingabeaufforderung können in diesem Terminal alle Shell Anweisungen benutzt werden.

- Für eine Navigation zwischen den Systempfaden und für eine schnelle Verwaltung der Ordner wird ein *Dateimanager*[44] vorausgesetzt. Der Befehl

  ```
  sudo apt-get install pcmanfm
  ```

  richtet den „*PcManFm*“ Manager auf dem System ein, der diese Aufgaben übernimmt.

- „**gedit**“ ist ein Texteditor, der die Bearbeitung von Textdateien möglich macht. Darüber hinaus können damit C-Quelltexte erstellt oder bearbeitet werden, die die Befehlssyntax farblich hervorheben. Dieser Editor ist durch den Befehl

  ```
  sudo apt-get install gedit
  ```

  erhältlich.

- Die Installation der Pakete „*obconf*“ und „*obmenu*“ mit

  ```
  sudo apt-get install obconf
  sudo apt-get install obmenu
  ```

  richten eine Art *Panel*[45] ein, mit dem der Aufruf aller genannten Applikation bewerkstelligt werden kann. Dabei übernimmt „*obmenu*“ dessen visuelle Darstellung und „*obconf*“ dessen Konfiguration.

[44] Visuelle Umgebung zur Verwaltung von Dateien
[45] Visuelle Desktopleiste mit ausführbaren Applikationen

- Ein „*gcc Compiler*“ wird für die Kompilierung von C-Quellcodes auf einem Linux System benötigt. Über den Aufruf

  ```
  sudo apt-get install build-essential
  ```

  im Terminal wird dieser installiert und kann folglich verwendet werden. Der „*gcc Compiler*“ spielt eine wichtige Rolle bei der Realisierung der anstehenden Aufgaben.

- Zum Schluss werden noch die Pakete „*xorg*“, „*xinit*“ und „*hal*“ mit den folgenden Aufrufen

  ```
  sudo apt-get install xorg
  sudo apt-get install xinit
  sudo apt-get install hal
  ```

  geholt und installiert. Diese stellen einige Funktionsbibliotheken für die grafische Bedienung der aufgelisteten Programme bereit.

4.1.4.3 Systemeinstellungen

Die Inbetriebnahme des Robotinos wird nur über einen Power Button vorgenommen, der sich auf der Kommandobrücke befindet. Aus dynamischen Gründen ist der dauerhafte Einsatz von Eingabegeräten, wie die Tastatur und die Maus, im späteren Verlauf nicht mehr möglich. Deswegen werden wiederholte Systemabläufe wie der *Login*[46], in soweit automatisiert, dass lokale Befehlseingaben nicht mehr benötigt werden. Herfür dienen die vorher installierten Programme, deren Verwendung ebenfalls vorkonfiguriert werden muss.

Diese Systemeinstellungen werden mit den nachfolgenden Punkten durchgeführt:

- Als erstes müssen die installierten Programme so eingestellt werden, dass die komplette Systemkonfiguration mit deren Hilfe vorgenommen werden kann. Nach dem Anmelden im System mit den gegebenen Benutzerdaten, steht dem Benutzer die Shell zur Verfügung. Mit der Eingabe des Befehls

  ```
  sudo startx
  ```

  in der Shell wird der installierte Fenstermanager (openbox) gestartet. In diesem Fenster kann durch die rechte Maustaste ein „*Steuermenü*“ (obmenu) geöffnet werden, aus dem vordefinierte Programme durch Klicken gestartet werden können (siehe Abbildung 4.17). Dieses Menü soll den Aufruf von dem Terminal (xterm), dem Dateimanager (pcmanfm) und der Konfiguration dieses Menüs (obmenu) beinhalten. Dazu wird das Terminal durch Drücken auf den Eintrag „*Terminal emulator*“ geöffnet:

[46] Systemanmeldung

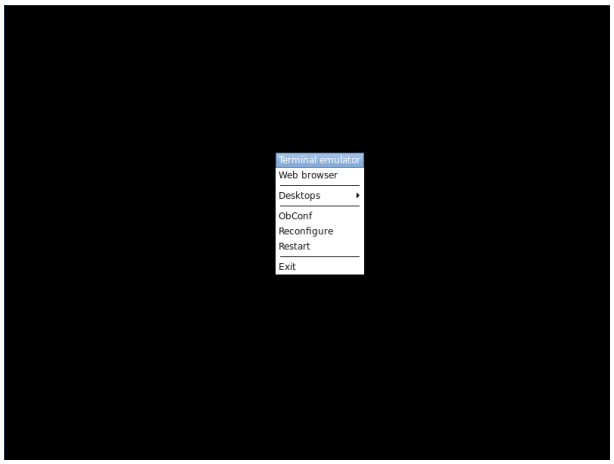

Abbildung 4.17: Menüaufruf unter dem Fenstermanager Openbox

Durch die Eingabe des Befehls in das geöffnete Terminal

```
sudo obmenu
```

wird das Konfigurationsfenster des Obmenüs aufgerufen. An diesem Beispiel können bei Bedarf auch alle anderen Programme gestartet werden. In dem neu geöffneten Fenster, in der Abbildung 4.18 dargestellt, werden nun oben die vorherigen Einträge sichtbar. Diese können übers Markieren mit der Maus im unteren Teil des Fensters bearbeitet werden. Beispielsweise wird „*Terminal emulator*“ markiert, in dem Feld „*Label*“ in „*Terminal*“ umbenannt und für den Aufruf von xterm im „*Execute*“ Feld „*xterm*“ eingetragen. Nicht benötigte Einträge können so auch gelöscht oder zusätzliche hinzugefügt werden.

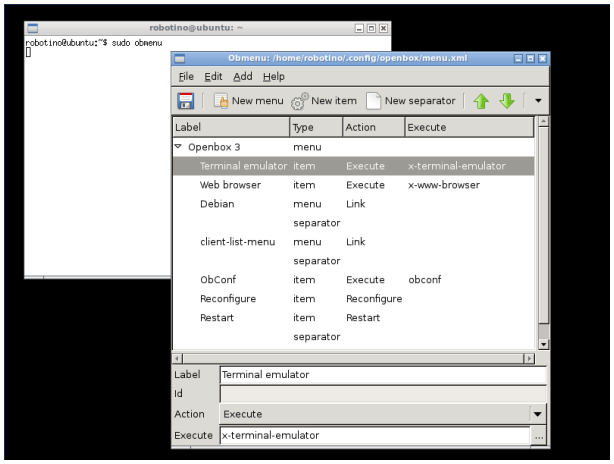

Abbildung 4.18: Konfigurationsfenster der Menüeinträge

Nach diesem Prinzip wird das Menü mit den oben genannten Programmen vervollständigt und gespeichert. Leider bietet dieses Verfahren nicht die Option die Programme mit „*sudo*“ zu starten. Aus diesem Grund können unberechtigte Aktionen,

wie beispielsweise das Bearbeiten von Systemdateien über den Dateimanager, nicht abgewickelt werden. Um dies jedoch machen zu können, werden die Applikationen einfach über das Terminal mit „*sudo*“ gestartet.

- Eine manuelle Eingabe der Benutzerdaten nach dem Systemstart wird mit einem Autologin umgangen. Für dieses Ziel wird eine Systemdatei bearbeitet, in die zusätzliche Befehle eingetragen werden. Dafür können jetzt die installierten Werkzeuge eingesetzt werden. Das verantwortliche *Skript*[47] befindet sich in dem Systempfad

  /etc/init/tty1.conf

  und hat den folgenden Inhalt

```
1  # tty1 -getty
2  #
3  # This service maintains a getty on tty1 from the point the system is
4  # started until it is shut down again.
5
6  start on stopped rc RUNLEVEL=[2345]
7  stop on runlevel [!2345]
8
9  respawn
10 exec /sbin/getty -8 38400 tty1
```

  Für den automatischen Login muss lediglich die letzte Zeile durch den folgenden Befehl

```
10 exec /bin/login -f robotino < /dev/tty1 > /dev/tty1 2>&1
```

  ersetzt und anschließend die Änderung im Skript abgespeichert werden. Dadurch ist ein Autologin des Benutzers „*robotino*“ bei jedem Start gewährleistet.

- Während des Starts von Ubuntu wird der GRUB Loader ausgeführt, in dem zusätzliche Bootoptionen ausgewählt werden können. Für diese Auswahl stehen dem Benutzer standardgemäß zehn Sekunden zur Verfügung, in denen er seine Entscheidung treffen muss. Falls dies innerhalb der zehn Sekunden nicht geschieht, wird das System im normalen Zustand gestartet. Aufgrund der fehlenden Eingabegeräte am Robotino, muss bei jeder Inbetriebnahme diese Zeit abgewartet werden. Die Manipulation des zuständigen Skripts verkürtzt diese Zeit und führt zu einem beschleunigten Startvorgang. Für diesen Zweck ist die Datei

  /boot/grub/grub.cfg

[47] Kommandozeilenprogramm

verantwortlich mit dem folgenden Ausschnitt

```
50 ...
51 if [ ${recordfail} = 1]; then
52     set timeout=-1
53 else
54     set timeout=10
55 fi
56 ...
```

Der Befehl „*set timeout=10*“ in der Zeile 54 definiert die Auswahlzeit. Dieser wird auf „*1*“ gesetzt und anschließend die Änderung in dem Skript gespeichert. Beim nächsten Start des Robotinos wird nun eine Sekunde gewartet, bis der standardmäßige Bootvorgang ohne jegliche manuelle Bestätigung fortgesetzt wird. Bei Bedarf können in den nachfolgenden Zeilen dieses Skripts auch zusätzliche Bootoptionen hinzugefügt, entfernt oder auch deren Bezeichnungen geändert werden.

- Nach dem beschleunigten Systemstart und dem Autologin kann der Aufruf des „*openbox*“ Fenstermanagers ebenfalls automatisiert werden. In diesem Fall muss dafür eine versteckte Datei aus dem Benutzerordner „*robotino*“

  ```
  /home/robotino/.bashrc
  ```

  manipuliert werden. Versteckte Dateien sind an einem Punkt vor dem Dateinamen erkennbar und lassen sich über das Terminal mit dem Befehl

  ```
  ls -a
  ```

  anzeigen. Um diese Datei bearbeiten zu können, muss über das Terminals zu dem robotino Ordner navigiert und dann dort die „*.bashrc*“ Datei mit dem gedit Editor und den root Rechten mit

  ```
  sudo gedit .bashrc
  ```

  geöffnet werden. Das Ende der Datei wird mit den Befehlszeilen

```
99  ...
100 if [ $(tty) == „/dev/tty1“ ]; then
101     startx
102 fi
```

ergänzt und abgespeichert. Mit diesem Ablauf wird der Fenstermanager bei jedem Bootvorgang automatisch gestartet und für weiterführende Einsätze bereitgestellt.

Hinweis: Entstehende Fehler in den Skripten können zum Stopp des Bootvorgangs führen. Um dem entgegenzuwirken, können mit der Tastenkombination Alt +Strg+F2 alternative Logineingaben geöffnet und über die manuell fortgefahren werden.

- Die fehlende Batterie des PC/104, die für die Energieversorgung des BIOS benötigt wird, setzt die Systemzeit des Betriebssystems bei jedem Ausschalten des Robotinos zurück. Dieses Zurücksetzen der Uhr wird von Ubuntu registriert und daraufhin eine langanhaltende Systemüberprüfung durchgeführt. Eine Konfiguration der Zeit, während des Starvorgangs, verhindert diese Überprüfung. Dazu wird ein eigen erstelltes Skript in den Startvorgang eingebunden. Die Befehlseingabe

  ```
  sudo gedit /etc/init.d/runtime
  ```

  im Terminal erzeugt eine neue Datei mit dem Namen „*runtime*“. Diese wird mit dem Inhalt

  ```
  1 date -s „20/06/2010 12:00:00“
  ```

  gefüllt und gespeichert. Das eingesetzte Datum sollte später sein, als der Zeitpunkt der Installation von Ubuntu Lucid Lynx. Diese Datei wird anschließend mit den Befehlen

  ```
  sudo chmod +x /etc/init.d/runtime
  sudo update-rc.d runtime defaults
  ```

  in den Ablauf des Startvorgangs aufgenommen und jedes Mal ausgeführt. Mit diesen Mitteln wird immer eine neue Systemzeit gesetzt und eine verzögernde Überprüfung verhindert.

Mit den durchgeführten Schritten wurde letztendlich Ubuntu Lucid Lynx auf der CF Speicherkarte installiert und für den Einsatz auf dem Robotino konfiguriert.

4.1.5 Systemtest

Nach der Fertigstellung der Installation kann nun die CF Speicherkarte in den Robotino reingesteckt und der automatische Systemstart durchgeführt werden. Wenn bei diesem Fortgang keinerlei Fehler entstehen und der Fenstermanager openbox selbstständig gestartet wird, war die Installation erfolgreich. Für weiterführende Tests wird die Systemauslastung angeschaut und ein einführendes C-Programm „Hallo Welt!“ geschrieben und ausgeführt.

4.1.5.1 Systemauslastung

Die Auslastung des Systems kann über drei wesentliche Merkmale nach der Installation von Ubuntu überprüft werden. Zu diesen Merkmalen gehört einmal der benutzte Speicherplatz auf der CF Speicherkarte, die Auslastung des Arbeitsspeichers und des Prozessors. Diese Tests sollten Sinngemäß auf dem Robotino stattfinden.

- Der Speicherplatzbedarf des Betriebssystem lässt sich über das Terminal mit dem Befehl

  ```
  sudo df -h
  ```

  anzeigen. Hieraus lässt sich erkennen, dass zurzeit ca. 30% des 4 GB großen Speichers von dem Betriebssystem verwendet werden. Es gibt also noch genug Platz für weitere Applikationen.

- Mit der Eingabe des Befehls

  ```
  sudo top
  ```

  im Terminal wird eine Ausführliche Auslastung des Prozessors angezeigt. In dem momentanen Zustand wird 7% des 500 MHz Prozessors verwendet. Daher ist auch hier eine Reserve für weitere Prozesse vorhanden.

- Der Auslastung des Arbeitsspeichers des PC/104 wird mit dem Befehl

  ```
  sudo free -m
  ```

  aufgerufen. In einem nicht belasteten Zustand werden von dem gesamten 256 MB Arbeitsspeicher, 86 MB benutzt. Dies entspricht einer Auslastung von ca. 10%.

4.1.5.2 Einführendes C-Programm

Ein C-Quellcode wird mit einem Compiler verarbeitet und kann dann über das Terminal ausgeführt werden. In den vorrangegangenen Schritten wurde bereits ein gcc Compiler für diese Aufgabe auf dem Betriebssystem installiert. Mit dem Aufruf

```
aptitude show gcc
```

kann die Version und der Status dieser Software nochmal überprüft werden. Für den endgültigen Funktionstest des Compilers wird ein einführendes C-Programm „Hallo Welt“ geschrieben und ausgeführt. Es wird eine neue Datei mit dem Namen „*hallo.c*“ in den Ordner

```
/home/robotino/
```

erstellt und mit dem C-Quelltext

```
1 #include <stdio.h>
2
3 int main (void){
4     printf(„Hallo Welt! \n“);
5 }
```

versehen und gespeichert. Als nächstes wird die Datei über das Terminal kompiliert und anschließend ausgeführt. Standartmäßig wird dies über die Befehle

```
sudo gcc hallo.c
sudo ./a.out
```

verrichtet. Wenn im Terminal keine Fehlermeldungen angezeigt werden und die Ausgabe „*Hallo Welt*“ erscheint, wurde der Quellcode vom Compiler erfolgreich interpretiert und der Test im positiven Sinne beendet.

Hinweis zum Compiler: Zusätzlich können weiterführende Optionen verwendet werden, die unter anderem den Namen der ausführenden Datei definieren oder auch zusätzliche Verlinkungen zu externen Bibliotheken beinhalten können. Die nachfolgende Zusammenstellung beschreibt die wesentlichen Parameter:

-o Hinter dem Zeichen wird der Name der Ausgabedatei neu definiert (Beispiel: sudo gcc hallo.c -o test)

-c Kompilierung des Quellcodes ohne einer Verlinkung

-O Mit diesem Aufruf wird der Quellcode mit einer Gewichtung 1-3 optimiert

-l Anbindung zusätzlicher Bibliotheken an den zu kompilierenden Quellcode

-ansi Es werden nur ANSI C-konforme Konstrukte erlaubt

-Wall Aktivierung von gcc Compiler unterstützten Warnungen

-pedantic Ausgabe aller Warnungen und Fehlermeldungen

-v Ausgabe der aktuellen Kommandos und Schritte des Compilers

-g Einfügen von Debugging Symbolen in die Binärdatei

Der gcc Compiler verfügt über eine bequemere Methode den Aufruf mehrerer Optionen zusammenzufassen. Dafür wird eine „*makefile*“ Datei angelegt, die die Anleitung der Kompilierung beschreibt. Insofern muss bei einer komplexeren Quellcodestruktur nur das Makefile aufgerufen werden. Auf die manuelle Eingabe zusätzlicher Informationen zur Kompilierung kann dabei verzichtet werden.

Bei der Kompilierung der Quellcodes achtet der Compiler auf die Änderungszeit der Ausgabedatei. Falls die Ausgabedatei aktuell ist, werden keine Änderungen durchgeführt. Bei der Rücksetzung der Uhrzeit des PC/104 beim Ein- und Ausschalten des Robotinos, kann dies zu Problemen bei Rekompilierungen führen. Auf diese Komplikationen muss in den weiteren Schritten gesondert geachtet werden.

4.1.6 Nützliche Hinweise

Einige zusätzliche Programme können eine erneute Installation und die zugehörige Konfiguration des Ubuntu Betriebssystems verhindern. Darüber hinaus erleichtern sie die Arbeit mit der minimalen Version des Betriebssystems. Es ist zu empfehlen, parallel eine Komplettversion von Ubuntu auf einem Leistungsstärkeren Rechner als den PC/104 zu installieren und bei Bedarf Vergleiche zu der Robotino Version zu ziehen.

gparted Mit der Installation des Programms gparted auf der Vollversion von Ubuntu können Partitionen der minimalen Installation auf der CF Speicherkarte nachträglich geändert werden. Damit kann eine 4 GB Installation auf 8 GB erweitert oder auch verkleinert werden.

cheese Das Programm cheese auf der Vollversion ist eine Applikation, mit der eine angeschlossene Webcam an dem PC aktiviert und die Aufnahme angezeigt werden kann. Dadurch wird die Funktion der Kamera und die Verfügbarkeit der Linux Unterstützung ersichtbar.

image Weiterhin kann über das Terminal, sowohl auf der Voll- wie auch auf der Minimalversion, ein Image der jeweiligen Systeme erstellt oder auf einen anderen Datenträger kopiert werden. Ein Image ist eine binäre Datei aus der, im umgekehrten Fall, das Betriebssystem wiederhergestellt werden kann. Der dafür verantwortliche Befehl

```
sudo dd if=Quelle of=Ziel <Optionen>
```

kann über das Terminal ohne zusätzliche Installationen aufgerufen werden. Das Ziel und die Quelle beschreiben die Speichermedien für diesen Prozess, beispielsweise „*/dev/hda*“, „*/dev/sdb*“ und so weiter. Ebenso können die Destinationen in Form einer Imagedatei „*of= /robotino.img*“ verwendet werden. Der Einsatz von Optionen ist nicht zwingend notwendig, daher werden sie nicht weiter erläutert. Mit diesem Hintergrund kann die minimale Installation von Ubuntu mit den zugehörigen Konfigurationen gesichert und auf erneute, zeitaufwendige Installation verzichtet werden.

4.2 Kommunikation

Die meisten Server laufen aus Stabilitätsgründen mit einem Linux Betriebssystem. Oft ist es so, dass die Server sich nicht vor Ort befinden und bei anstehenden Aktualisierungen ferngesteuert werden müssen. Die weite Verbreitung des Windows Betriebssystem in privaten Haushalten führt zu einer Methode, diese Server mit einem Windows Rechner über ein Netzwerk zu bedienen. Diese Technik kann auch auf den Robotino übertragen werden und dadurch lokale Eingaben der Tastatur oder der Maus entbehrlich machen. Hierbei werden jedoch weitere Software und zugehörige Konfigurationen auf der Host- sowie der Clientseite benötigt. Der Host ist in diesem Fall der Robotino mit dem Linux Betriebssystem und der Client ist ein Windows Rechner, der über ein Netzwerk sich mit dem Host verbindet. In

diesem Abschnitt wird dieses Gefüge mit den zugehörigen Komponenten und Einstellungen erklärt und realisiert.

4.2.1 Technische Komponenten

Zu den technischen Komponenten gehören einmal der Robotino selbst, ein Windows Rechner und ein Access Point, dass eine Drahtlose Übertragung sicherstellt. Auf dem Robotino befindet sich das Linux Betriebssystem, das über den Windows Rechner bedient werden soll. Der Robotino ist in dieser Anordnung der Host (Gastgeber) und der Windows Rechner der Client (Benutzer). Die komplette Verbindung wird über die drahtlose Netzwerkkarte des Windows Rechners mit dem Access Point hergestellt, das wiederum über ein LAN Kabel am Robotino angeschlossen ist. Das Access Point wird damit als eine Art Tauschstation eingesetzt, zu dem beide Seiten eine Verbindung aufbauen müssen, um Daten von einer Seite zur anderen leiten zu können. Da es sich hierbei um eine Netzwerkverbindung handelt, geschieht die Adressierung über eine IP, die bei jedem Gerät individuell sein muss. Die Adressen des Robotinos und des Access Points können frei gewählt werden, allerdings müssen sie sich nur in der letzten Zahl unterscheiden, sonst wird ein Verbindungsaufbau zueinander nicht möglich. Aus praktischen Gründen wird daher die bislang geltende IP des Robotinos „*192.169.0.13*“ beibehalten und die IP „*192.168.0.2*“ für das Access Point gewählt. Zusammengefasst wird diese Struktur nochmal in der Abbildung 4.19 verdeutlicht.

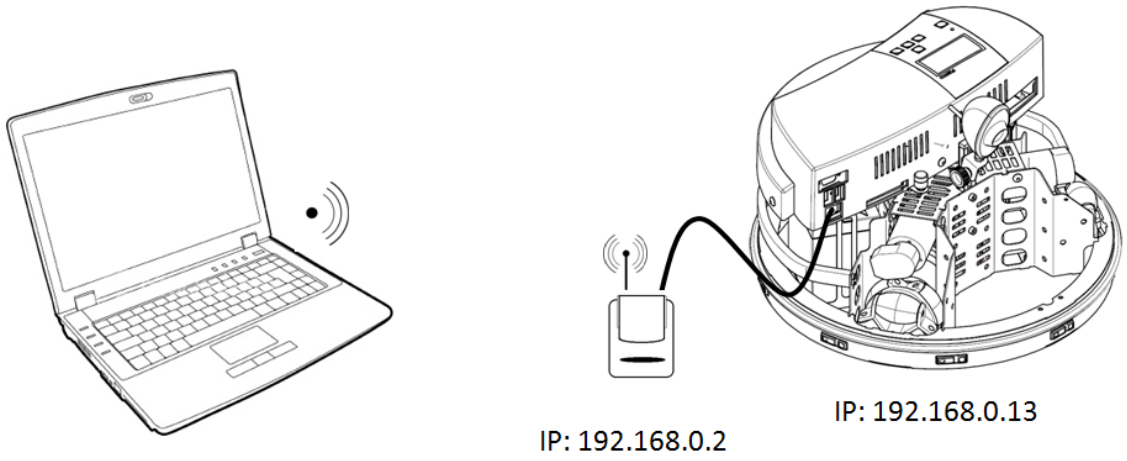

Abbildung 4.19: Komponenten der Fernsteuerung (Istockphoto , Zugriff: 19.04.2010)

4.2.2 Access Point Konfiguration

Für eine erfolgreiche Verbindung des Access Points zum Clint und zum Host müssen einige Einstellungen des Geräts vorgenommen werden. Das Access Point ist für mehrere Funktionen ausgelegt, die auf der Rückseite umgeschaltet werden können. Für die Fernsteuerung

wird die Schalterstellung „*AP*“ verwendet. Die Funktionen „*Client*“ und „*RT*“ haben für die Fernsteuerung keine Bedeutung und daher werden diese nicht weiterhin vertieft. Die Einstellungen des Geräts werden mit dem Client Rechner durchgeführt, indem das Access Point mit einem LAN Kabel in der Schalterstellung „*AP*“ mit dem Rechner verbunden wird. Nun wird für eine erfolgreiche Kommunikation der beiden Komponenten eine Adressierung des Windows Computer vorgenommen. Im Bezug darauf müssen die Netzwerkeinstellungen von Windows geöffnet und dort die IP „*192.168.0.13*“ (Verbindungsaufbau wie beim Robotino) mit der Subnetzmaske „*255.255.255.0*“ eingetragen werden. Dieser Kommunikationsaufbau kann nur dann gelingen, wenn sich die IP der Geräte nur in der letzten Ziffer unterscheiden. Falls die IP des Access Points nicht bekannt ist, kann ein manueller Reset mit Hilfe eines Knopfes auf dem Gerät durchgeführt werden. Danach können die Standardeinstellungen benutzt werden, die in der Bedienungsanleitung aufgeführt sind. Nachdem das erledigt ist, kann das Konfigurationsfenster des Access Points über die Eingabe der IP „*192.168.0.2*“ in einem Explorer aufgerufen werden. Dabei sind die Logindaten

Benutzername: **admin**

Passwort: **airlive**

zu verwenden. Bei einer erfolgreichen Anmeldung an dem Access Point werden letztendlich folgende Einstellungen vorgenommen:

- In der Abbildung 4.20 werden unter dem Reiter „*Basic Settings*“ die Grundeinstellungen des Access Points durchgeführt. Mit der Wahl der SSID “*Robotino Linux*“ wurde der Name ausgesucht, der dann in der Liste der Drahtlosen Netzwerkverbindungen angezeigt werden soll. Der Channel wird auf „*11*“ gesetzt, damit er sich von den umgebenden WLAN Netzen unterscheidet und eine stabilere Übertragung gewährleistet. Da es sich hierbei nicht um eine risikoreiche Übertragung handelt, kann auf die Sicherheitseinstellungen verzichtet werden. Zum Speichern müssen noch die Änderungen akzeptiert werden.

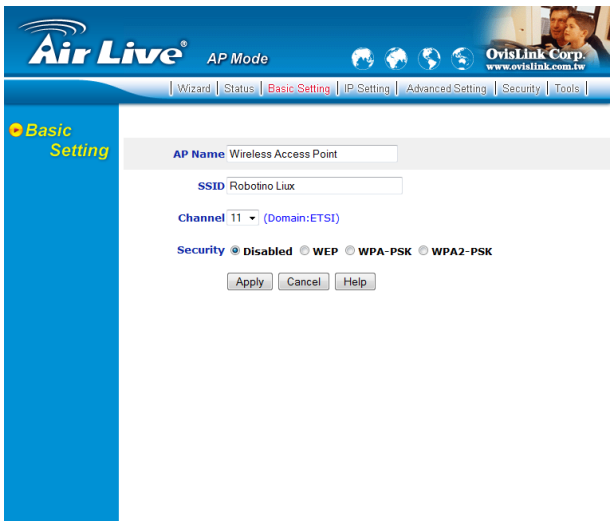

Abbildung 4.20: Grundeinstellungen des Access Points

- Als letzten Punkt der Konfiguration muss die IP Einstellung des Geräts vorgenommen werden (siehe Abbildung 4.21). Dies wird unter dem Reiter „*IP Settings*“ erledigt. Unter diesem Punkt wird eine feste IP „*192.168.0.2*“ mit einer Subnetzmaske „*255.255.255.0*“ gesetzt. Der DHCP Server wird eingeschaltet und eine IP Bandbreite von „*192.168.0.100*“ bis „*192.168.0.199*“ definiert. Das Gateway und der DNS Server können auf „*0.0.0.0*“ bleiben. Abschließend wird die Änderung übernommen und die Konfiguration beendet.

Abbildung 4.21: IP Einstellung des Access Points

Damit ist die Konfiguration des Access Points fertiggestellt. Des Weiteren wird das Gerät am Robotino angeschlossen und mit den zugehörigen Einstellungen des Linux Betriebssystem fortgefahren.

4.2.3 Linux IP Konfiguration

Die Fertigstellung des Access Points erlaubt dessen Verbindung mit dem Linux Betriebssystem. Dazu wird jetzt die IP Konfiguration von Ubuntu vorgenommen. Dies sollte direkt auf dem Robotino durchgeführt werden, da bei dieser Methode eine Identifizierung der Netzwerkkarte des PC/104 stattfinden wird. Durch die Eingabe des Befehls

```
ifconfig
```

im Terminal des Linux Betriebssystems, werden Informationen über aktive Netzwerkschnittstellen ausgegeben. Das Ziel ist es in Erfahrung zu bringen, ob es sich bei der Schnittstelle um „*eth0*[48]“ oder „*eth1*“ handelt.

Hinweis: Unterschiedliche Modelle des Robotinos können unter Umständen andere Schnittstellenbelegungen haben. Bei Kommunikationsproblemen sollte zuerst darauf geachtet werden.

[48] Nummerierung der Netzwerkadapter

Ein anderer Weg die Verfügbarkeit der Hardware zu testen, ist die Benutzung der Befehle

```
dmesg | grep eth0
dmesg | grep eth1
```

Diese können auch inaktive Netzwerkschnittstellen identifizieren. Nachdem dies erledigt ist, werden die Netzwerkeinstellungen des Systems vorgenommen. Die zugehörige Skriptdatei ist unter dem Pfad

```
/etc/network/interfaces
```

zu finden. Dieses Skript beinhaltet standardgemäß den folgenden Inhalt

```
1  # This file describes the network interfaces available on your system
2  # and how to activate them. For more information, see interfaces(5).
3
4  # The loopback network interface
5  auto lo
6  interface lo inet loopback
7
8  # The primary network interface
9  auto eth0
10 iface eth0 inet dhcp
```

In dieser Datei wird schließlich eine feste IP für den Robotino definiert, indem die letzten Zeilen auskommentiert und mit dem folgenden Einträgen erweitert werden

```
1  # This file describes the network interfaces available on your system
2  # and how to activate them. For more information, see interfaces(5).
3
4  # The loopback network interface
5  auto lo
6  interface lo inet loopback
7
8  # The primary network interface
9  # auto eth0
10 # iface eth0 inet dhcp
11
12 auto eth1
13 iface eth1 inet static
14 address 192.168.0.13
15 netmask 255.255.255.0
```

An dieser Stelle wird die beabsichtigte IP des Robotinos verwendet und die zuvor identifizierte Netzwerkschnittstelle eingesetzt. Im Anschluss der Fertigstellung wird das Skript im System mit dem Befehl

```
sudo /etc/init.d/networking restart
```

neu geladen und bei einer fehlerlosen Konfiguration eine Verbindung vom Robotino zum Access Point hergestellt.

4.2.4 Programme zur Fernsteuerung

Für die grundsätzliche Fernsteuerung des Betriebssystems auf dem Robotino werden zusätzliche Programme auf dem Host sowie auf dem Client benötigt.

4.2.4.1 Linux SSH

Unter Linux wird zunächst mit dem Befehl

```
sudo apt-get install ssh
```

eine weitere Applikation heruntergeladen und installiert. Dieses Programm dient zur Nutzung der Linux Shell von einem entfernten Rechner über ein Netzwerk. Die Kommunikation kann bei Bedarf verschlüsselt werden und den Datenaustausch sichern. Diese Anwendung bedarf keiner weiteren Einstellungen. Der Robotino ist damit bereit ein Kommunikationskanal mit einem entfernten Rechner herzustellen.

4.2.4.2 Windows PuTTY

Auf dem Windows Rechner müssen ebenfalls zwei nachträgliche Anwendungen installiert und konfiguriert werden. Das Programm PuTTY, erhältlich unter dem Link

Download: `http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html`

stellt Clientseitig eine Verbindung mit dem Host her und bietet dem Anwender unter Windows ein grafisches Fenster, in dem die Shell Befehle geschrieben und auf den Host übertragen werden können. Andersherum werden die Ausgaben in der Shell vom Host zurück gesendet und angezeigt. Dieses Programm wird mit den herkömmlichen Mitteln unter Windows installiert und für den Einsatz mit dem Robotino vorbereitet. Zuerst werden die Grundeinstellungen in dem „*Session*“ Menü in den Angriff genommen. Dort wird unter dem Host Namen die IP „*192.168.0.13*“ des Robotinos eingetragen. Der Port kann auf „*22*“ und der Verbindungstyp auf SSH bleiben (siehe Abbildung 4.22).

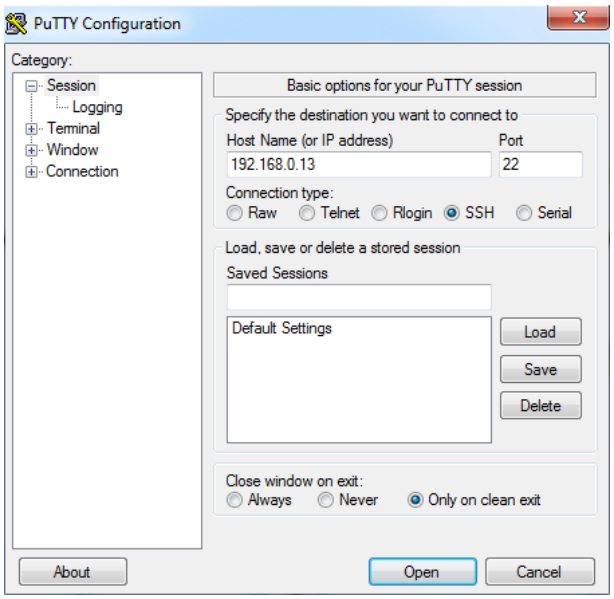

Abbildung 4.22: PuTTY Grundeinstellungen

Unter dem Menü „*Window->Translation*“ sollte in dem *Drop Down Fenster*[49] „*UTF-8*“ gesetzt sein. Dies dient zur korrekten Ausgabe von individuellen Zeichen wie „ä“, „ö“ und so weiter. Siehe dazu die die nachfolgende Abbildung 4.23:

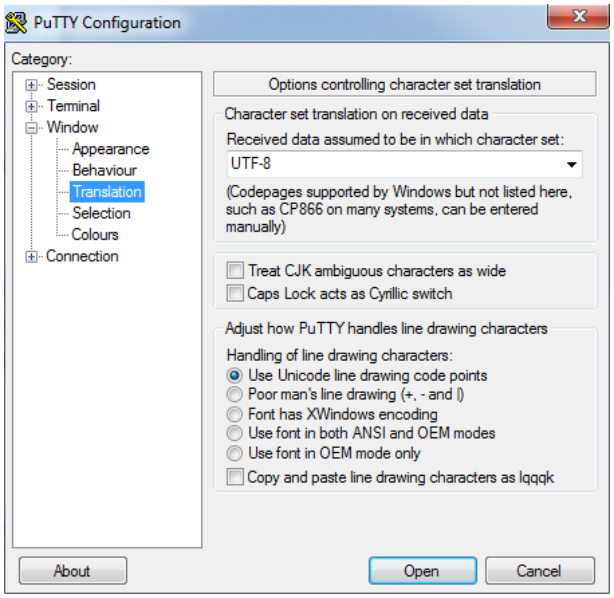

Abbildung 4.23: PuTTY Übersetzung

Unter dem Menüpunkt „*Connection->Data*“ wird der zugehörige Loginname „*robotino*“ eingesetzt, der bei der Anmeldung mit dem Linux Betriebssystem benutzt wird (siehe Abbildung 4.24).

[49] Ein nach unten herausfallendes Auswahlmenü

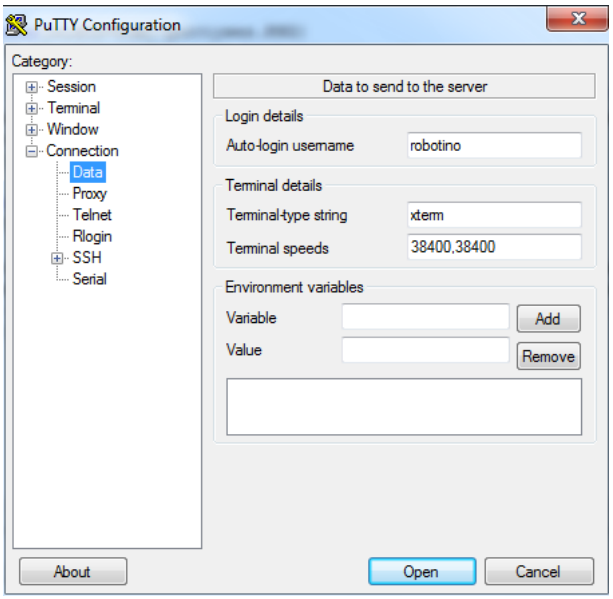

Abbildung 4.24: PuTTY Anmeldename

Zum Schluss sollten die Konfigurationen unter den Grundeinstellungen von PuTTY gespeichert werden, um erneute Eingaben bei wiederholtem Kommunikationsaufbau zu ersparen.

4.2.4.3 Windows WinSCP

Mit PuTTY alleine können nur die Shell Befehle übertragen werden. Für den Versand von Daten vom Client zum Robotino wird ein zweites Programm WinSCP angewendet, das unter dem Link

Download: http://www.chip.de/downloads/WinSCP_13007380.html

heruntergeladen werden kann. WinSCP stellt bei der Anwendung ein grafisches Fenster mit zwei Abschnitten bereit, wobei die linke Seite die Ordner des Clients beinhaltet und die rechte, die Ordner des Robotinos. Mit der Drag & Drop Funktion können Daten von einem System zum andren kopiert werden. Dieses Programm wird ebenfalls mit gewöhnlichen Mitteln auf dem Client installiert und ausgeführt. Bei dieser Anwendung wird für die Kommunikation mit dem Robotino nur die „*Sitzung*“ bearbeitet (siehe Abbildung 4.25).

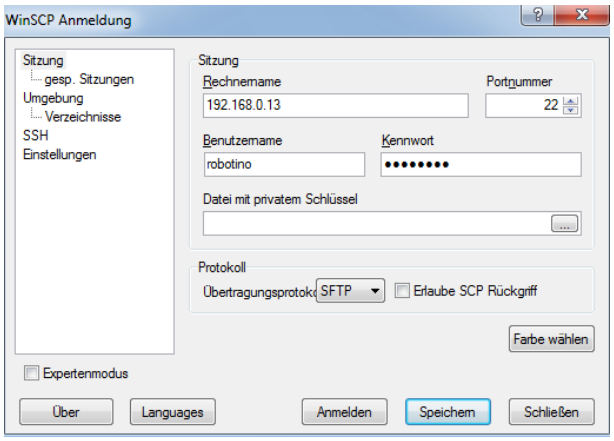

Abbildung 4.25: WinSCP Grundeinstellungen

In dem obigen Fenster wird, ähnlich wie bei PuTTY, die IP „*192.168.0.13*“ des Robotinos für den Rechnernamen und der Port „*22*“ verwendet. Weiterhin werden die Logindaten „*robotino*“ für den Benutzernamen und für das Passwort benutzt. Für die Datenübertragung ist das „*SFTP*“ Protokoll zuständig. Anschließend können die Einstellungen ebenfalls gespeichert und bei wiederholten Aufrufen schneller geöffnet werden.

4.2.5 Verbindungsaufbau

Die Installation und Konfiguration für die Fernsteuerung des Robotinos wurde damit abgeschlossen und es kann über den Client eine Kommunikation zum Host hergestellt werden. Zuerst wird eine Verbindung von dem Windows Rechner über die WLAN Karte zum dem Access Point aufgebaut. Über die Netzwerkeinstellungen werden die verfügbaren, drahtlosen Netzwerke aufgerufen und dort die Verbindung mit dem vordefinierten Namen „*Robotino Linux*“ hergestellt. Diese ist nur dann Verfügbar, wenn der Robotino eingeschaltet ist und sich in der Reichweite vom lokalen WLAN befindet. Für die Fernsteuerung des Robotinos können jetzt nacheinander die Programme PuTTY und WinSCP mit den erstellten Profilen gestartet werden. Mit dieser Software ist das Linux Betriebssystem des Robotinos von einem Windows Rechner komplett steuerbar.

4.3 EIA-232 Ansteuerung

Für die Ansteuerung der Sensoren und Aktoren des Robotinos muss das I/O Board über die serielle Schnittstelle des PC/104 angesprochen werden. Wie fast bei jedem Betriebssystem gibt es auch unter Ubuntu eine Möglichkeit dies zu machen. Zu diesem Zweck wird unter Linux ein C-Code geschrieben, in dem die dafür zugeschnittene Bibliothek „*termios*“ verwendet wird. Termios stellt für die Programmierung der EIA-232 Schnittstelle Funktionen bereit, die die Kommunikation mit dem I/O Board erlaubt und dementsprechend die Grundlage für die Ansteuerung des Robotinos bildet. In diesem Abschnitt werden diese

Funktionen vorgestellt und eine komplette Kommunikation der Aktoren und Sensoren des Robotinos realisiert.

4.3.1 Einbindung von Termios

Termios ist bei der minimalen Ubuntu Installation bereits integriert und kann mit dem Aufruf

```
1 #include <termios.h>
```

in jedem C-Quellcode eingebunden werden. Zusätzlich werden noch weitere Header gebraucht, wie beispielsweise <stdio.h>, <fcntl.h> und <unistd.h>, die die Werkzeugpalette vervollständigen. Ein einfaches Beispiel zur Generierung eines C-Programms wurde bereits gezeigt. Termios stellt die Struktur bereit

```
1 struct termios {
2     c_iflag; /*Eingangs Flags*/
3     c_oflag; /*Ausgangs Flags*/
4     c_cflag; /*Kontroll Flags*/
5     c_lflag; /*Lokale Flags*/
6     c_cc[NCCS]; /*Spezielle Zeichen*/
7     c_ispeed; /*Eingang Baud Rate*/
8     c_ospeed; /*Ausgang Baud Rate*/
9 }
```

mit der fast alle Einstellungen der Übertragungsparametern der seriellen Übertragung vorgenommen werden können. Die restlichen Eigenschaften werden beim Öffnen der Schnittstelle zugewiesen.

4.3.2 Öffnen/Schließen der Schnittstelle

Das Öffnen und Schließen der seriellen Schnittstelle unter Linux lässt sich ähnlich wie bei einer Datei durchführen. Dabei wird eine Variable deklariert, die auf die jeweilige Schnittstelle mit dem Befehl „*open*“ zugewiesen wird. Mit dieser Variablen werden im Nachhinein die Einstellungen zu den Übertragungsparametern vorgenommen. Vor dem Öffnen einer Schnittstelle müssen jedoch einige Informationen berücksichtigt werden. Darunter sind Optionen die besagen, ob von der Schnittstelle nur gelesen oder auch geschrieben werden soll. Ein Aufruf der seriellen Schnittstelle kann wie folgt aussehen

```
1 int fd_ser;
2 fd_ser=open(„/dev/ttyS0“, O_RDWR | O_NOCTTY | O_NDELAY);
3 if (fd_ser == -1) {
4     perror(„Unable to open /dev/ttyS0 - „);
5     return 1;
6 }
```

Folgende Optionen können beim Öffnen der Schnittstellen verwendet werden:

/dev/ttyS: /dev/ttyS bildet die eigentliche serielle Schnittstelle ab. Unter Linux werden alle Schnittstellen unter dem Ordner

```
/dev/
```

als Datei angezeigt und können von dort aus geöffnet werden. Auf einem Computer befinden sich meist mehrere Schnittstellen, die von 0 bis maximal 5 definiert und hinter dem Aufruf gesetzt werden. Bei diesem Beispiel wird die erste serielle Schnittstelle "'*/dev/ttyS0*“ des Rechners geöffnet.

O_RDONLY: Schnittstelle wird nur zum Lesen geöffnet.

O_WRONLY: Schnittstelle wird nur zum Schreiben geöffnet.

O_RDWR: Schnittstelle wird zum Lesen und Schreiben geöffnet.

O_NOCTTY: Wenn sich der Aufruf auf ein Terminal bezieht, so wird der Prozess nicht mehr über dieses gesteuert werden können.

O_NDELAY: Aufruf wird ohne Blockierung geöffnet. Unter Umständen können dabei die Werte null Bytes beinhalten.

O_SYNC: Port wird im synchronen Modus geöffnet. Jeder Aufruf wird den Prozess solange aufhalten, bis dieser nicht zu Ende durchgeführt ist.

Falls die Schnittstelle nicht geöffnet werden kann, wird anhand des Rückgabewertes eine Fehlermeldung im Terminal ausgegeben und der Prozess abgebrochen.

Um die geöffnete Schnittstelle wieder zu schließen und die gesetzten Optionen zurückzusetzen, wird abschließend die zugehörige Kommandozeile

```
1 close(fd_ser);
```

in den Quelltext eingefügt.

4.3.3 Grundeinstellungen

Nach dem Öffnen der seriellen Schnittstelle können die Grundeinstellungen vorgenommen werden. Dabei werden die typischen Übertragungsparameter gesetzt, wie die Baudrate oder die definierte Bitreihenfolge, die die Zusammenarbeit mit der Gegenkomponente arrangieren. Hierzu wird die Bibliothek termios mit den jeweiligen Strukturoptionen in den Quellcode eingefügt

```
1 struct termios options;
```

Mit Hilfe des zuvor gesetzten Zeigers auf die serielle Schnittstelle und den zwei wesentlichen Funktionen „*tcgetattr()*“ und „*tcsetattr()*“ können die Strukturelemente und die Eigenschaften der geöffneten Schnittstelle sinngemäß manipuliert werden. Hierfür können Elemente mit den folgenden Parametern definiert werden:

c_cflag: Mit diesem Strukturelement werden die Kontrolloptionen der zuvor geöffneten seriellen Schnittstelle gesetzt. Diese dürfen nicht direkt zugewiesen, sondern müssen mit den Bitoperatoren „*AND*“, „*OR*“ und „*NOT*“ verknüpft werden. Insgesamt sind dafür die Optionen aus der nachfolgenden Tabelle 4.1 vornehmbar:

Konstante	**Beschreibung**
CBAUD	Bitmaske für die Baudrate
B0	0 baud
B50	50 baud
B75	75 baud
B110	110 baud
B134	134 baud
B150	150 baud
B200	200 baud
B300	300 baud
B600	600 baud
B1200	1200 baud
B1800	1800 baud
B2400	2400 baud
B4800	4800 baud
B9600	9600 baud
B19200	19200 baud
B38400	38400 baud
B57600	57600 baud
B76800	76800 baud
B115200	115200 baud
EXTA	Externe Zeitrate
EXTB	Externe Zeitrate

CSIZE	Bitmaske für die Datenbits
CS5	5 Datenbist
CS6	6 Datenbist
CS7	7 Datenbist
CS8	8 Datenbist
CSTOPB	2 Stoppbits (sonst 1)
CREAD	Empfang aktivieren
PARENB	Paritätsbit aktivieren
PARODD	Ungerade Parität (sonst gerade)
HUPCL	Verbindungsabbruch bei Beendigung des letzten Prozesses
CLOCAL	Keine Benutzeränderung
LOBLK	Ausgabe wird geblockt, falls Prozess nicht aktuell ist
CNEW_RTSCTS CRTSCTS	Hardwarekontrolle aktivieren (wird nicht von jeder Plattform unterstützt)

Tabelle 4.1: Kontrolloptionen der seriellen Schnittstelle unter Linux (MKSSoftware , Zugriff: 17.04.2010)

Die Optionen beinhalten zwei Konstanten „*CLOCAL*“ und „*CREAD*“, die für jeden Einsatz der seriellen Schnittstelle empfohlen werden und wie folgt aussehen

```
1 options.c_cflag |= (CLOCAL | CREAD);
```

Diese Einstellungen bewirken eine Reibungslose Kommunikation und aktivieren den Empfang von ankommenden Daten.

Die Übertragungsgeschwindigkeit kann nicht nur über die Konstante „*CBAUD*“ definiert werden, sondern auch über zwei weitere Funktionen. Die Eingehende Baudrate kann beispielsweise mit den Aufrufen

```
1 cfgetispeed(&options, B115200); /*Eingangsbaudrate lesen*/
2 cfsetispeed(&options, B115200); /*Eingangsbaudrate setzen*/
```

oder auch nach dem gleichen Prinzip die Ausgehende Baudrate mit den Funktionen

```
1 cfgetospeed(&options, B115200); /*Ausgangsbaudrate lesen*/
2 cfsetospeed(&options, B115200); /*Ausgangsbaudrate setzen*/
```

gelesen oder gesetzt werden.

c_lflag: Die Lokalen Optionen werden mit „*c_lflag*“ festgelegt und beschreiben die Handhabung der eingegangenen Zeichen. Zu diesem Strukturelement gehört die nachfolgende Konfigurationsgruppe:

Konstante	Beschreibung
ISIG	Terminal-Sonderzeichen einschalten
ICANON	Zeilenorientierter Eingabemodus
XCASE	Umwandeln von eingegebenen Groß- und Kleinbuschstaben
ECHO	Einschalten der ECHO - Funktion
ECHOE	Gelöschte Zeichen mit Leerzeichen überschreiben
ECHOK	Zeichen löschen oder zur Neueingabe in neue Zeile Positionieren
ECHONL	Ausgabe von NL
NOFLSH	Abschalten des Leerens vom Puffer bei INTR oder QUIT
IEXTEN	Einschalten des erweiterten Zeichensatzes für Eingabe
ECHOCTL	Darstellung von Steuerzeichen als Zeichen
ECHOPRT	Ausgabe von gelöschten Zeichen für Hardcopy
ECHOKE	Zeichen beim Löschen einer Zeile entfernen
FLUSHO	Leeren des Ausgabepuffers
PENDIN	Neuausgabe von nichtverarbeitenden Eingabezeichen
TOSTOP	Senden von SIGTTOU bei Ausgabe durch Hintergrundprozesse

Tabelle 4.2: Lokale Optionen der seriellen Schnittstelle unter Linux (MKSSoftware , Zugriff: 17.04.2010)

Der größte Teil der möglichen Konfiguration wird für den Robotino nicht weiter verwendet. Um einen unverarbeiteten Datenempfang zu aktivieren, reicht lediglich die folgende Konfiguration

```
1 options.c_lflag &= (ICANON | ECHO | ECHOE | ISIG);
```

c_iflag: Mit diesem Element wird der Empfang von Zeichensätzen konfiguriert. Unter anderem kann der Paritätscheck aktiviert oder bestimmte, ankommende Systemzeichen umbenannt werden. Diese Konfigurationen werden beim Robotino nicht verwendet und daher nicht weiter erläutert.

c_oflag: Ähnlich wie beim c_iflag werden hierbei die Einstellungen zum Versand von Zeichensätzen getätigt. Es können ebenfalls Systemzeichen umbenannt oder bestimmte Zeitverzögerungen eingebaut werden. Diese Konfigurationen werden beim Robotino nicht verwendet und daher nicht weiter erläutert.

c_cc[NCCS]: Mit „*c_cc[NCCS]*“ können Steuerzeichen oder Timeout Parameter direkt gesetzt werden, mit denen beispielsweise eine serielle Kommunikation über eine bestimmte Tastenkombination abgebrochen werden kann. Diese Konfigurationen kommen beim Robotino nicht zum Einsatz und daher werden sie nicht weiter vertieft.

Nachdem die termios Struktur zweckmäßig bearbeitet worden ist, werden abschließend die Optionen an die Hardware mit der Funktion „*tcsetattr()*“ zugeteilt. Die Zuweisung verfügt über drei Eigenschaften, die in der Tabelle 4.3 aufgelistet sind.

Konstante	**Beschreibung**
TCSANOW	Änderungen werden sofort übernommen
TCSADRAIN	Vor den Änderungen werden momentane Prozesse abgewartet
TCSAFLUSH	Buffer wird gelehrt und Änderungen werden übernommen

Tabelle 4.3: Optionen zur Aktivierung der termios Konfiguration (MKSSoftware , Zugriff: 17.04.2010)

Für eine sofortige Übernahme der Änderungen wird folglich der Befehl

```
1 tcsetattr(fd_ser, TCSANOW, &options);
```

verwendet. Diese Konfigurationen gelten dann so lange, bis sie nicht umbearbeitet werden oder bis die geöffnete Schnittstelle nicht wieder geschlossen wird.

4.3.4 Senden und Empfangen von Daten

Das Senden und Empfangen von Daten über die serielle Schnittstelle kann nach einer korrekten Konfiguration verrichtet werden. Es werden zuerst Arrays im C-Programm definiert, die die jeweiligen Informationen zwischenspeichern sollen. Zum Versand werden die zu sendenden Daten in das zugehörgien Array eingefügt und das Ganze mit dem Funktionsaufruf

```
1 write(fd_ser, sCmd, 5); /*Sende 5 Bytes aus dem Array sCmd*/
```

in den Sendebuffer der geöffneten seriellen Schnittstelle kopiert und gesendet. Für den Empfang der Daten wird im umgekehrten Fall der Dateninhalt aus dem Empfangsbuffer mit dem Befehl

```
1 read(fd_ser, sResult, 5); /*Empfange 5 Bytes aus dem Buffer in das sResult Array*/
```

in das zugehörige Array geschrieben.

4.4 Webcam Ansteuerung

Für die Ansteuerung der USB Webcam unter Linux ist eine weitere Bibliothek namens *„video4linux“* zuständig, die zugleich den Treiber für die vorhandene Logitech Kamera bildet sowie eine Reihe von anderen Kameras unterstützt. Der gleichermaßen kontinuierliche Fortschritt der Webcamentwicklung führte im Laufe der Zeit zu einem immer größer werdenden Funktionsumfang der Hardware. Dieser Fortschritt wurde ebenfalls in der video4linux Bibliothek angeglichen. Für spezielle Funktionsaufrufe ist daher die weiterentwickelte Bibliothek *„video4linux2“* unumgänglich. Angesichts des komplexen Protokollaufbaus der USB Datenübertragung wird mit dieser Bibliotheken der eigentliche Zugriff auf die Rohdaten der Schnittstelle vermieden. Durch deren Anbindung an die Robotino Steuerung lässt sich folglich die Webcam ohne Probleme nutzen. Anhand dieses Beispiels können auch andere USB Devices für Steuer- und Regelzwecke verwendet werden. In diesem Abschnitt wird die Einrichtung der Bibliotheken und nur für die Aufgabenstellung in Frage kommenden Funktionsaufrufe beschrieben.

4.4.1 Einbindung der Bibliotheken

Die video4linux Bibliotheken sind im Ubuntu Linux Betriebssystem noch nicht vorhanden und müssen deswegen noch installiert werden. Dies lässt sich bei einer stehenden Internetverbindung durch die Eingabe des Befehls

```
sudo apt-get install libv4l-0 libv4l-dev
```

im Terminal bewerkstelligen. Nach der Fertigstellung können die Bibliotheken in jeden geschriebenen C-Quellcode mit den Einträgen

```
1 #include <libv4l1.h>
2 #include <linux/videodev.h>
3 #include <libv4l2.h>
4 #include <linux/videodev2.h>
```

im Header einfügt und im aktuellen Programm verwendet werden. Da es sich dabei um nachinstallierte Funktionen handelt, müssen diese bei jedem Kompiliervorgang angeknüpft werden. Am Beispiel des Hallo Welt Programms lässt sich das mit dem Aufruf

```
sudo gcc hallo.c -lv4l1 -lv4l2
```

realisieren.

4.4.2 Öffnen/Schließen der Webcam

Das Öffnen und Schließen der USB Webcam unter Linux wird ähnlich wie bei der seriellen Schnittstelle durchgeführt. Dabei wird eine weitere Variable deklariert, die auf die jeweilige Kamera mit dem Befehl „*v4l1_open*“ zugewiesen wird. Im Unterschied zu seriellen Schnittstelle wird hier die vorher installierte Bibliothek beim Öffnen der Kamera verwendet. Mit dieser Variablen können im Nachhinein die Einstellungen zu der Aufnahme der Webcam vorgenommen sowie die aufgenommen Bilder gelesen werden. Ein Aufruf kann wie folgt aussehen

```
1 int fd;
2 fd=v4l1_open(„/dev/video0“, O_RDONLY);
3 if (fd == -1) {
4     perror(„Unable to open webcam device„);
5     return 1;
6 }
```

Auf die gleiche Weise wie bei der seriellen Schnittstelle wird auch in diesem Fall eine stellvertretende Datei aus dem Ordner

```
/dev/
```

geöffnet. Für die Kameras wird der Aufruf *„video“* benutzt. An dieser Stelle kann ebenfalls zwischen 64 angeschlossenen Kameras gewählt werden, indem hinter dem Aufruf eine Nummer von 0 bis 63 gesetzt wird. Falls die Webcam nicht geöffnet werden kann, wird anhand des Rückgabewertes eine Fehlermeldung im Terminal ausgegeben und der Prozess abgebrochen.

Um die geöffnete Webcam wieder zu schließen, wird abschließend die zugehörige Kommandozeile

```
1 v4l1_close(fd);
```

in den Quelltext eingefügt.

4.4.3 Video4linux Funktionen

Video4linux stellt eine Reihe von Funktionen und Strukturen bereit, über die die wesentlichen Einstellungen der Kameras vorgenommen werden können. Die Befehle werden aus der vorher deklarierten Variablen *„fd“*, einem ioctl Aufruf und einer zugehörigen Struktur gebildet. Die folgenden Punkte beschreiben einige dieser Aufrufe und deren Eigenschaften:

- Die Eigenschaften der angeschlossen Kamera werden durch den „*VIDIOCGCAP*“ ioct Aufruf angefordert. Die zugehörige „*video_capability*“ Struktur beinhaltet im Einzelnen folgende Elemente:

```
1  struct video_capability {
2      name[32]
3      type
4      channel
5      audios
6      maxwidth
7      maxheight
8      minwidth
9      minheight
10 }
```

name[32]: In diesem Element wird der Produktname der geöffneten Kamera abgelegt. Der Name kann bis zu 32 Zeichen enthalten.

type: Dieses Element beinhaltet die Eigenheit der angeschlossenen Kamera. Unter anderem Zeigt dieses Feld, ob die aufgenommenen Daten direkt in den Speicher geladen werden können oder ob beispielsweise eine schwarzweiße Aufnahme möglich ist.

channels: Falls die Kamera unterschiedliche Aufnahmekanäle hat, wird hier deren Anzahl dargestellt.

audios: Die meisten Webcams haben ein Mikrofon integriert, damit neben der Videoübertragung auch eine Sprachübertragung möglich ist. In diesem Zusammenhang wird hierbei festgehalten, ob ein Audiokanal verfügbar ist.

maxwidth und maxheight: Die maximale Auflösung der Kamera in die Breite und Höhe wird in diesen Feldern festgehalten.

minwidth und minheight: Auf die gleiche Weise wie die maximale Breite und Höhe, ist auch die minimale definiert. Zwischen diesen Angaben können noch weitere Auflösungen gewählt werden.

- Mit Hilfe des Aufrufs „*VIDIOCGWIN*“ werden Informationen zur Bildgröße angefordert. Mit „*VIDIOCSWIN*“ können die Eigenschaften der Bildgröße gesetzt werden. Diese Aufrufe verwenden die „*video_window*“ Struktur, die im Einzelnen folgende Elemente aufweist:

```
struct video_window {
    x
    y
    width
    height
    chromakey
    flags
    clips
    clipcount
}
```

width: Anhand dieses Elements kann die Breite der Bildaufnahme in Pixel gesetzt werden. Hierbei muss die minimale und maximale Breite aus „*VIDIOCGCAP*“ berücksichtigt werden.

height: Anhand dieses Elements kann die Höhe der Bildaufnahme in Pixel gesetzt werden. Hierbei muss die minimale und maximale Höhe aus „*VIDIOCGCAP*“ berücksichtigt werden.

Die restlichen Elemente dieser Struktur sind für die Aufgabenstellung nicht relevant und werden daher nicht näher beschrieben. Die Auswahl der Auflösung ist nur in bestimmten Stufen möglich, die in der Tabelle 3.8 entnommen werden können. Mit höherer Auflösung steigt auch der Speicherbedarf und diesbetreffend sinkt die Geschwindigkeit der Bildaufnahme.

Hinweis: Damit die gesetzten Einstellungen aktiv werden, müssen sie noch einmal im Programmverlauf eingelesen werden.

- „*VIDIOCGPICT*“ liefert Informationen zu den Bildeigenschaften. Mit „*VIDIOCSPICT*“ können die Eigenschaften der Bildaufnahme mit der Skalierung von 0 bis 65535 gesetzt werden. Diese Einstellungen spielen eine große Rolle bei der Qualität der Bilder und auch bei der Erkennung von Mustern für die anstehende Regelung. Sie werden in der „*video_picture*“ Struktur zusammengefasst, die im Einzelnen folgende Elemente aufweist:

```
struct video_picture {
    brightness
    hue
    colour
    contrast
    whiteness
    depth
    palette
}
```

brightness: Brightness variiert die Helligkeit der Bildaufnahme. Beispielsweise können bei einer dunklen Aufnahmeumgebung alle aufgenommenen Pixelwerte um einen bestimmten Wert erhöht werden. Auf diese Weise werden jedoch die Objekte nicht deutlicher sondern werden lediglich nur im Helligkeitsbereich verschoben.

hue: Mit hue wird der Farbton der Bilder definiert. Die Auswahl dieser Eigeneschaft bestimmt die Zusammensetzung der Farbe, sodass es nur bei der individuellen Empfindung des menschlichen Auges eine Rolle Spielt.

colour: Mit diesem Element wird die Intensität der einzelnen Grundfarben rot, grün, blau gesetzt. Durch die Erhöhung des Wertes werden diese drei Farben knalliger dargestellt.

contrast: Mit dem Kontrast wird die Intensität zwischen hellen und dunklen Bereichen eines Bildes erhöht.

whiteness: Mit whiteness wird die Blässe in einem schwarzweißen Bild gesteuert. Im Extremfall kann die Einstellung so hoch gestellt werden, dass keine tiefschwarzen Pixel mehr in einem Bild vorhanden sind.

depth: Ein Bild kann aus unterschiedlichen Bittiefen zusammengesetzt sein. Mit diesem Element kann diese Tiefe für eine Bildaufnahme gesetzt werden.

palette: Die Farbzusammensetzung kann anhand unterschiedlicher Farbräume definiert werden. Abhängig von der Webcam ist es mit diesem Element möglich, zwischen mehreren Farbraumdefinitionen zu wählen. Hierzu sind vordefinierte Schlüsselwörter einsetzbar. In der Tabelle 3.8 ist dargestellt, dass es sich bei der vorliegenden Kamera um eine 24 Bit True Color Kamera handelt. Damit ist für dieses Modell die Farbpalette *„VIDEO_PALETTE_RGB24“* maßgebend.

Alle Funktionsaufrufe wurden universell für verschiedene Kameras konstruiert. Ob die Einstellungen auch wirklich umgesetzt werden können, hängt immer von den Eigenschaften der jeweiligen Hardware ab.

4.4.4 Video4linux2 Funktionen

Video4linux2 ist wesentlich umfangreicher und komplexer aufgebaut, als der Vorgänger. Mit dieser Bibliothek können spezielle Funktionen gesteuert werden, die unter anderem die Belichtung und die Lichtverstärkung verändern können. Als Standardeinstellung der vorliegenden Kamera werden diese Optionen automatisch an die aktuellen Verhältnisse angepasst. Diese Einstellungen erschweren eine feste Justierung der Bildaufnahme und benötigen des Weiteren eine enorme Rechenzeit, sodass sich bei ständig wechselnden Lichtintensitäten die Bildrate bei einer vollen Auflösung von 15 Bildern pro Sekunde auf 5 reduziert wird. Aus diesem Grund wird auf die Funktionalitäten dieser Punkte eingegangen, um die entstehen Nebenwirkungen bei der Realisierung zu vermeiden.

- Die automatische Belichtungs- und Lichtverstärkungsregelung wird in der Video4linux2 Bibliothek in der Struktur *„V4L2_CID_EXPOSURE_AUTO“* gesetzt. Diese Struktur beinhaltet folgende Elemente

```
struct V4L2_CID_EXPOSURE_AUTO {
    V4L2_EXPOSURE_AUTO
    V4L2_EXPOSURE_MANUAL
    V4L2_EXPOSURE_SHUTTER_PRIORITY
    V4L2_EXPOSURE_APERTURE_PRIORITY
}
```

  Um die Automatik abzuschalten, muss das erste Element der Struktur *„V4L2 _EXPOSURE_AUTO“* mit dem Wert *„0“* belegt werden. Bei der Video4linux2 Bibliothek werden die Einstellungen über zwei ioctl Aufrufe und zwei weitere Strukturen durchgeführt. Deren Hintergrund ist nicht von weiterführender Bedeutung und daher als regelmäßigen Aufruf bei diesen Prozessen anzusehen. Als ein einführendes Beispiel wird im folgenden Code die Syntax zur deaktivierung der oben genannten Regelung gezeigt

```
memset (&qctrl, 0, sizeof(qctrl));
qctrl.id=V4L2_CID_EXPOSURE_AUTO;
v4l2_ioctl(*fd, VIDIOC_QUERYCTRL, &qctrl);
memset (&control, 0, sizeof(control));
control.id=V4L2_CID_EXPOSURE_AUTO;
control.value=0;
v4l2_ioctl(*fd, VIDIOC_S_CTRL, &control);
```

- Die Lichtverstärkung kann nach der Abschaltung der Automatik manuell über eine weitere Struktur gesetzt werden. Dazu wird ein Wert zwischen 1 und 255 an *„V4L2_CID_GAIN“* nach dem oben genannten Beispielprinzip übergeben.
- Als letzte nützliche Funktion zur Steuerung der Webcam kann ebenfalls manuell die Belichtung mit einem Wert von 1 bis 10000 mit *„V4L2_CID _EXPOSURE_ABSOLUTE“* variiert werden.

Diese Funktionen runden die Steuerbarkeit der vorliegenden Kamera ab und können bei der Regelung des Robotinos verwendet werden. Für eine detailliertere Beschreibung der Steuerfunktionen kann bei Bedarf die Spezifikation der genannten Bibliotheken im Internet eingesehen werden.

4.4.5 Einlesen der Kamerabilder

Aus der Beschreibung der vorliegenden Kamera geht hervor, dass sie eine 24 Bit True Color Aufnahme unterstützt. Die dreidimensionale Aufteilung der Grundfarben in einem digitalen Bild führt an dieser Stelle dazu, dass jedes Pixel in den jeweiligen Ebenen durch ein Byte repräsentiert wird. Ein Bild besteht folglich bei voller Auflösung aus 640x480x3 Bytes, die bei jedem Leseprozess an eine Programmvariable derselben Speichergröße übergeben werden müssen. Dies lässt sich bei der Video4linux Bibliothek mit dem Aufruf

```
1 v4l1_read(fd, picture, 640x480x3);
```

bewerkstelligen. Bei diesem Vorgehen ist „*fd*“ ein Zeiger, der auf die Webcamschnittstelle gerichtet ist. Bei diesem Einlesen der Bilder liegen die einzelnen Farbebenen hintereinander in der Reihenfolge blau, grün und rot in einem eindimensionalen Array der Variablen „*picture*“. Diese können bei Bedarf in die gewohnte dreidimensionale Ebene in der regulären Reihenfolge rot, grün und blau (RGB) aufgeteilt und im weiteren Verlauf des Programms genutzt werden.

4.4.6 Speichern der Kamerabilder

Die eingelesen Bilder können als Datei gespeichert und dann visuell geöffnet werden. Für diesen Zweck spielt das Speicherformat und die Farbdarstellung eine Rolle, denn diese werden im Header jeder Bilddatei definiert und von dem ausführenden Programm vor dem Öffnen interpretiert. Wie bereits beim technischen Hintergrund einer Webcam erwähnt wurde, ist Portable PixMap die einfachste Variante für dieses Vorhaben. Für ein SW Bild ist nach dem gleichen Prinzip Portbale GreyMap verwendbar.

4.4.6.1 Farbbild

Für die Speicherung eines Farbbildes muss im Programmverlauf eine neue, binär beschreibbare Datei mit der Endung „*.ppm*“ geöffnet werden. In die ersten drei Zeilen dieser Datei wird der zugehörige Header eingefügt. Der Header für ein Bild mit der höchsten Kameraauflösung kann wie folgt aussehen:

```
1 P6
2 640 480
3 255
```

P6: In der ersten Zeile des Headers befindet sich die Identifikationsnummer. Anhand dieser Nummer wird ausgesagt, dass es sich um drei Werte (rot, grün und blau) pro Pixel handelt, die jeweils ein Byte lang sind.

640 480: In der folgenden Zeile wird die Auflösung des Bildes definiert. Damit wird dem ausführenden Programm die Speicherlänge der Bilddatei mitgeteilt sowie die Zusammensetzung der Pixel.

255: In der letzten Zeile des Headers wird der dezimale Maximalwert eines Farbwertes festgelegt. In diesem Beispiel enthält die Bilddatei Farbwerte von 0 bis 255.

Im Anschluss an den Header folgen die eigentlichen Farbwerte, die durch ein Leerzeichen getrennt nach der RGB Reihenfolge, hintereinander geschrieben werden. Nach der Fertigstellung des Speichervorgangs kann das Farbbild visuell geöffnet werden.

4.4.6.2 SW Bild

Für die Speicherung eines SW Bildes muss im Programmverlauf ebenfalls eine neue, binär beschreibbare Datei geöffnet werden. Im Unterschied zum Farbbild, wird bei diesem Vorgehen die Endung „*.pgm*“ und eine andere Identifikationsnummer benutzt. Der Header für ein SW Bild mit der höchsten Kameraauflösung kann wie folgt aussehen:

```
1 P5
2 640 480
3 255
```

P5: Anhand dieser Identifikationsnummer wird ausgesagt, dass es sich um einen Grauwert pro Pixel handelt, der jeweils ein Byte lang ist. Ein SW Bild besteht im Gegensatz zum Farbbild aus zwei Dimensionen.

640 480: An der Auflösung des Bildes wird im Bezug auf das Farbbild nichts verändert.

255: In der letzten Zeile des Headers wird der dezimale Maximalwert des Grauwerts festgelegt. In diesem Beispiel enthält die Bilddatei Grauwerte von 0 bis 255.

Im Anschluss an den Header folgen die eigentlichen Grauwerte, die durch ein Leerzeichen getrennt, hintereinander geschrieben werden. Nach der Fertigstellung des Speichervorgangs kann das SW Bild visuell geöffnet werden.

4.5 RTW Embedded Coder Konfiguration

Die Installation von Linux auf dem Robotino bietet die Möglichkeit einen von dem Matlab/-Simulink Real-Time Workshop Embedded Coder erzeugten C-Code, der auf einer Windows Plattform erstellt wurde, auf dem Robotino zu kompilieren und anschließend auszuführen. Für eine Plattformunabhängige Kompilierung der C-Codes dieser Toolbox wird eine „*TMF*“ Datei verwendet, in der die Eigenheiten des Zielsystems definiert werden. Diese Datei wird für die anstehende Anwendung auf dem Robotino angepasst. Matlab/Simulink beinhaltet bereits standardgemäß eine „*ert_unix.tmf*“ Datei, die für den Einsatz unter Linux gedacht

ist. Diese kann aber nur dann für den RTW Embedded Coder verwendet werden, wenn auch Matlab auf der gleichen Plattform installiert ist. Aus diesem Grund wird die existierende TMF-Datei als Grundbasis verwendet und für den Gebrauch dieser Ausgangslage manipuliert. In diesem Abschnitt werden die dafür benötigten Schritte durchgeführt und anschließend von dem RTW Embedded Coder erstellte C-Codes eines einfachen Simulinkmodells auf dem Robotino kompiliert und ausgeführt.

4.5.1 Matlab/Simulink Konfiguration

Die RTW Embedded Coder Toolbox ist ein optionales Paket und sollte vor dem Gebrauch mit Matlab/Simulink auf einem Windows Rechner installiert sein. Nach dem Start von Matlab wird zuerst ein neues Simulinkmodell über „*File* → *New* → *Model*“ erstellt und im Bezug auf die Zielsetzung als „*robotino.mdl*“ beim späteren Abspeichern benannt. Die Benennung sollte für die Realisierung der Aufgabenstellung dieser Diplomarbeit beibehalten werden, da der Name einen Systempfad bildet wird, der bei der Kompilierung verfolgt wird. Die Wahl eines anderen Namens sollte bei der weiteren Konfiguration berücksichtigt werden. In dem neu geöffneten Simulinkmodell wird der Button für den „*Library Browser*“ aus der Menüleiste angeklickt, aus der Liste der Bibliotheken „*Real-Time Workshop Embedded Coder* → *Configuration Wizards*“ geöffnet und den „*ERT (optimized for floating-point)*“ Block in das Robotino Simulinkmodell per drag & drop übertragen (siehe Abbildung 4.26). Im Anschluss darauf wird durch einen Doppelklick auf diesen Block das Robotino Simulinkmodell für den Gebrauch des RTW Embedded Coders vordefiniert.

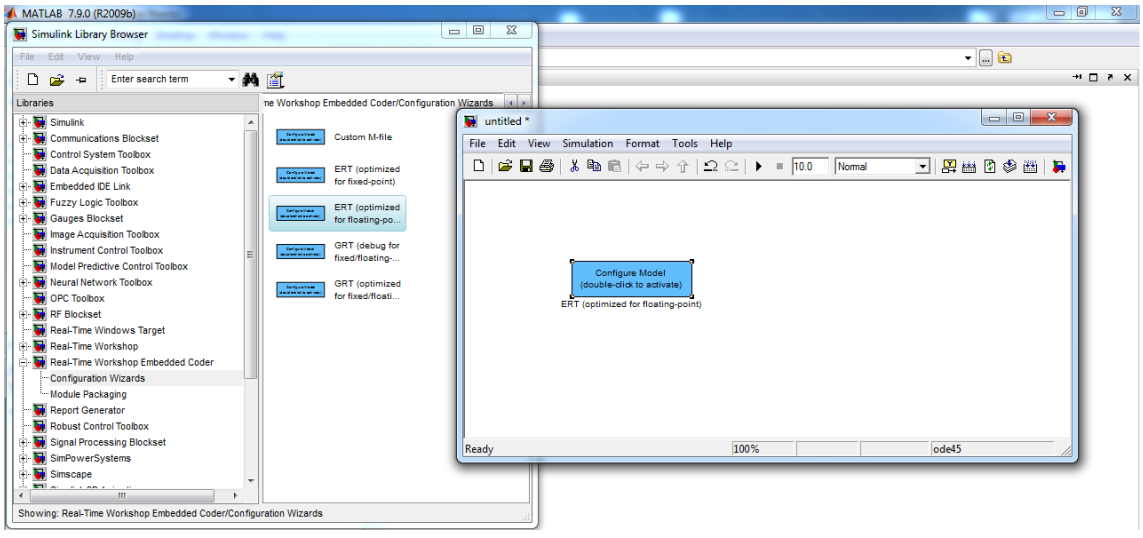

Abbildung 4.26: Voreinstellung des RTW Embedded Coders

Nach der automatischen Generierung der Konfigurationen kann der dafür benutzte Block wieder aus dem Modell entfernt werden. Einige weitere Einstellungen müssen dennoch manuell vorgenommen werden. Aus der Menüleiste von Simulink wird „*Simulation* → *Configu-*

ration Parametrs“ aufgerufen und der Eintrag „*Solver*“ links aus dem Baummenü markiert. Es sind folgende Einstellungen in der rechten Bildschirmhälfte vorzunehmen:

- **Type:** Fixed-step
- **Solver:** discrete (no continuous states)

Ebenso wird aus dem Baummenü „*Real-Time Workschop*“ markiert und die folgende Änderungen vorgenommen:

- **Template makefile:** robotino_ert_rtw.tmf
- **Generate code only:** Checkbox aktivieren

Zum Schluss kann noch die Unterfunktion „*Real-Time Workschop* → *Interface*“ aufgerufen und zur Reduzierung von Fehlermeldungen beim kompilieren der C-Codes auf dem Robotino die folgende Auswahl gesetzt werden:

- **Target function library:** C99 (ISO)

4.5.2 Robotino TMF-Datei

Bei der Erzeugung der C-Codes aus den Simulinkmodellen durch den RTW Embedded Coders wird automatisch ein „*makefile*“ erstellt, in dem Schritte und Dateipfade der späteren Kompilierung dieser Codes aufgeführt werden. Diese Schritte und Dateipfade müssen sich in Abhängigkeit der Zielsysteme unterscheiden. Die automatische Erzeugung dieser makefiles wird von den TMF-Dateien gesteuert. Dementsprechend wird eine eigen erstellte „*robotino_ert_rtw.tmf*“ Datei konstruiert, die ein passendes makefile für den Robotino generiert. Hierzu bildet eine bereits existierende Datei aus dem folgenden Matlab Installationspfad

```
C:\Program Files\MATLAB\R2009b\rtw\c\ert\ert_unix.tmf
```

die Basis. Diese Datei wird anhand der folgenden Anleitung verändert:

1. Es wird eine Kopie von der oben genannten Datei auf den Desktop gemacht, diese in „*robotino_ert_rtw.tmf*“ umbenannt und mit einem Windows Texteditor geöffnet.
2. Nach der „*SYS_TARGET_FILE = any*“ Zeile wird ein neuer Eintrag „*MAKEFILE _FILESEP = /*“ hinzugefügt

```
47 ...
48 SYS_TARGET_FILE = any
49 MAKEFILE_FILESEP = /
50 COMPILER_TOOL_CHAIN = unix
51 ...
```

3. Ab dem Eintrag „*ifeq ($(OPT_OPTS),$(DEFAULT_OPT_OPTS))*“ Zeile 278 bis 291 müssen alle if Anweisungen entfernt werden und die else Befehle übrig bleiben

```
275 ...
276 LIBS =
277 |>START_PRECOMP_LIBRARIES<|
278 LIBS += |>EXPAND_LIBRARY_NAME<|.a
279 |>END_PRECOMP_LIBRARIES<| |>START_EXPAND_LIBRARIES<|
280 LIBS += |>EXPAND_LIBRARY_NAME<|.a |>END_EXPAND_LIBRARIES<|
281 LIBS += $(S_FUNCTIONS_LIB) $(INSTRUMENT_LIBS)
282 ...
```

4. In der Zeile 276 unter „*LIBS =*“ werden die externen Bibliotheken für die Webcam eingebunden

```
275 ...
276 LIBS =-lv4l1 -lv4l2
277 ...
```

   Somit können die Webcamfunktionen im Zusammenhang mit dem RTW Embedded Coder genutzt werden.

5. Des Weiteren muss die neue Betriebssystemumgebung des Robotinos angepasst werden. Hierfür ist „*MATLAB_ROOT*“, „*ALT_MATLAB_ROOT*“, „*START_DIR*“ und „*COMPUTER*“ zuständig

```
91  ...
92  MATLAB_ROOT = /home/robotino/matlab/
93  ALT_MATLAB_ROOT = /home/robotino/matlab/
94  ...
95  START_DIR = /home/robotino/robotino_ert_rtw/
96  ...
97  ...
98  ...
99  ...
100 COMPUTER = GLNX86
101 ...
```

6. In der Zeile 211 wird noch einmal ein Pfad aus dem Robotino Betriebssystem gesetzt und die folgenden zwei Zeilen entfernt

```
210 ...
211 ADD_INCLUDES = /home/robotino/robotino_ert_rtw
212
213 ...
```

7. Damit ist die Konfiguration der TMF-Datei abgeschlossen. Zum Schluss wird die Datei gespeichert und in den Systempfad

   ```
   C:\Program Files\MATLAB\R2009b\rtw\c\ert\robotino_ert_rtw.tmf
   ```

   zurück geschoben.

Die TMF-Datei wurde so konfiguriert, dass bei der Erstellung der C-Codes aus dem Simulinkmodell automatisch ein makefile erzeugt wird, welches die passenden Regeln und Abläufe für die Kompilierung auf dem Robotino beinhaltet.

4.5.3 Zusätzliche Matlab Funktionen

Der RTW Embedded Coder bezieht sich bei der Kompilierung der C-Codes auf weitere Funktionen, die die Installation von Matlab enthält. Diese befinden sich jedoch nicht auf einem anderen Zielsystem und müssen von daher manuell auf den Robotino kopiert werden. Der Pfad für diese Funktionen wurde in dem vorherigen Abschnitt „*Robotino TMF-Datei*“ unter dem Punkt fünf definiert. Somit muss auf dem Robotino ein neuer Ordner „*matlab*“ unter

```
/home/robotino/
```

erstellt und die fehlenden Funktionen aus der Matlab Windows Installation

```
C:\Program Files\MATLAB\R2009b\
```

hinzu kopiert werden. Dabei handelt es sich um die folgenden drei Verzeichnisse:

- extern
- rtw
- simulink

Der RTW Embedded Coder wurde so konfiguriert, dass er C-Codes aus den Simulinkmodellen erstellen kann, die dann auf dem Robotino kompiliert und ausgeführt werden können.

4.5.4 Testmodell

Für die Erprobung der Wirksamkeit des erstellten Systemaufbaus wird ein einfaches Simulinkmodell erstellt und auf dem Robotino ausgeführt. Zu diesem Zweck wird das vorkonfigurierte Simulinkmodell „*robotino.mdl*“ geöffnet und ein Eingangs- und Ausgangsport aus dem Library-Browser hineingezogen. Beide Ports werden anschließend miteinander verbunden (siehe Abbildung 4.27).

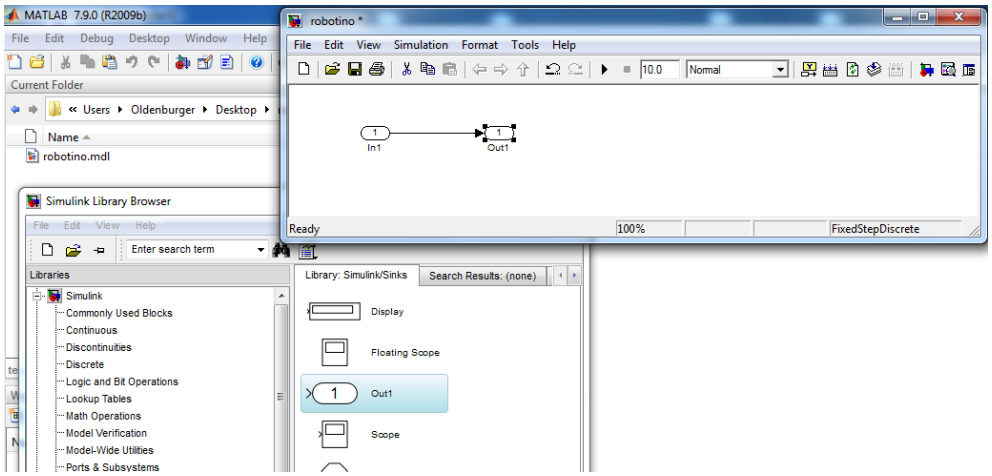

Abbildung 4.27: Testmodell für den Systemaufbau

Das erstelle Modell wird mit Hilfe des „*Incremental build*“ Buttons aus der Menüleiste des Simulinkfensters in einen C-Code gewandelt und in einem automatisch erstellten Ordner „*robotino_ert_rtw*“ zusammengefasst. Bei einer erfolgreichen Umwandlung werden keine Fehlermeldungen angezeigt und zum Schluss des Prozesses ein „*RTW Report*“ geöffnet. In dem RTW Report werden die Ergebnisse der RTW Embedded Coder Kompilierung zusammengefasst. Unter anderem können dort die eigentlichen erstellten C-Codes eingesehen werden oder die Definition aller Ein- und Ausgangsports. Damit letztendlich das Ganze auf dem Robotino ausgeführt werden kann, wird der erstellte Ordner „*robotino_ert_rtw*“ auf den Robotino in das

```
/home/robotino/
```

Verzeichnis kopiert. Weiterhin wird über das Terminal in den kopierten Ordner navigiert und zum Kompilieren der C-Codes der folgende Befehl

```
sudo make -f robotino.mk
```

eingegeben. Der Linux Compiler erstellt bei diesem Vorgang, anhand des „*robotino.mk*“ makefiles, eine ausführbare Datei in einem überliegenden Verzeichnis, die mit einem abschließenden Befehl

```
sudo ../robotino
```

ausgeführt werden kann. Bei einem erfolgreichen Abschluss wird in dem Terminal der Hinweis

„*Warning: The simulation will run forever.*
Generated ERT main won't simulate model step behavior.
To change this behavior select the 'MAT-file logging' option.“

eingeblendet. Um den Programmablauf zu stoppen, kann mit der Tastenkombination „*STRG + C*“ der Prozess beendet werden. Mit diesen Schritten ist das System des Robotinos für eine Realisierung der Steuerung und Regelung mit dem Matlab/Simulink RTW Embedded Coder bereit.

4.5.5 RTW Embedded Coder Schnittstelle

Eine Schnittstelle des RTW Embedded Coders mit externen Programmen des Linux Systems des Robotinos kann über die Datei „*ert_main.c*“ hergestellt werden. Diese Datei wird bei der Umwandlung des Simulinkmodells in einem leeren Zustand mit erzeugt. Der Inhalt dieser Datei ohne Kommentare sieht wie folgt aus

```
#include <stdio.h>
#include „robotino.h“
#include „rtwtypes.h“
static boolean_T OverrunFlag = 0;
void rt_OneStep(void) {
    if (OverrunFlag++) {
        rtmSetErrorStatus(rtM, „Overrun“);
        return;
    }
    robotino_step();
    OverrunFlag–;
}
int_T main(int_T argc, const char_T *argv[]) {
    robotino_initialize();

    printf(„Warning: The simulation will run forever.“
        „Generated ERT main won't simulate model step behavior.“
        „To change this behavior select the 'MAT-file logging' option.\n“);
    fflush((NULL));
    while (rtmGetErrorStatus(rtM) == (NULL)) {
    }
    return 0;
}
```

Über diese Datei werden in diesem Fall die Bibliothek für die Ansteuerung der seriellen Schnittstelle sowie die Bibliotheken der Webcam an das Simulinkmodell über den Header angebunden. Das ert_main.c Programm ist in zwei Funktionen „*rt_OneStep*“ und „*int_T main*“ aufgeteilt. Da es sich bei dem Simulinkmodell um einen Zeitdiskreten Ablauf handelt, definiert die Funktion rt_OneStep die einzelnen Schritte. In der int_T main Funktion werden die Komponenten des Simulinkmodells in der while Schleife abgearbeitet. Nach jedem Durchlauf sollte zweckmäßig die Funktion „*robotino_step()*“ aufgerufen werden, um

mit einer zeitfortgeschrittenen Abarbeitung des Simulinkmodells fortzufahren. Zusätzlich können hier die Ports (siehe Abbildung 4.27) für den Input mit rtU.In1 und für den Output rtY.Out1 eingesetzt und verwendet werden. Die einzelnen Definitionen der Ports und deren Speichergrößen können bei einer erfolgreichen Umwandlung des Simulinkmodells in dem RTW Report unter Code Interface Report eingesehen werden. Die Initialisierung von zusätzlichen Variablen kann vor der „*robotino_initialize()*“ Funktion vorgenommen werden.

4.5.5.1 Beispiel

Als ein einführendes Beispiel für die Anwendung der ert_main.c Datei wird anhand des Simulink Testmodells (siehe Abbildung 4.27) ein Programmablauf geschrieben, der den Ausgang nach jeden Schritt um einen Integerwert erhöht und im Terminal ausgibt. Dazu wird die folgende ert_main.c Datei verwendet

```
1  #include <stdio.h>
2  #include „robotino.h“
3  #include „rtwtypes.h“
4  static boolean_T OverrunFlag = 0;
5  void rt_OneStep(void) {
6      if (OverrunFlag++) {
7          rtmSetErrorStatus(rtM, „Overrun“);
8          return;
9      }
10     robotino_step();
11     OverrunFlag–;
12 }
13 int_T main(int_T argc, const char_T *argv[]) {
14     int start=1;
15     robotino_initialize();
16     fflush((NULL));
17     while (rtmGetErrorStatus(rtM) == (NULL)) {
18         start=rtY.Out1+1;
19         rtU.In1=start;
20         printf(„Ausgang: %f\n“, rtY.Out1);
21         robotino_step();
22     }
23     return 0;
24 }
```

Hinweis: Bei jeder Umwandlung der Simulinkmodelle mit dem RTW Embedded Coder wird die ert_main.c Datei neu erstellt und alle vorgenommenen Änderungen gehen dabei verloren. In Hinsicht darauf sollte immer eine Kopie dieser Datei außerhalb

des Entwicklungsordners von Matlab angelegt und bei Bedarf wieder zurück kopiert werden.

5 Realisierung

Nach dem Systemaufbau sind alle Komponenten zur Realisierung der Steuerung der Hardware und der Bahnregelung des Robotinos gegeben. In diesem Kapitel wird zu Beginn eine komplette Schnittstellenumgebung des Robotinos in einem Simulinkmodell erstellt und über die ert_main.c Datei mit den Linux Hardwareansteuerungen verbunden. Bei der Steuerung müssen noch einige Punkte bezüglich der Ausführzeit berücksichtigt und getestet werden. Für eine Bahnregelung wird noch ein visueller Bezugspunkt entwickelt, mit dem die genaue Position des Roboters und deren Ausrichtung berechnet werden kann. Für die Positionsbestimmung werden die von der USB Webcam aufgenommenen Bilder digital verarbeitet und die resultierenden Informationen in den Regelverlauf eingebunden. Die Bahnregelung wird in drei verschiedenen Formen realisiert und mit jeweils der zugehörigen Simulation verglichen. Die manuelle Eingabe der Ladeprozesse wird über Skriptdateien automatisiert, sodass nur wenige Schritte zum Start der Regelfahrt durchgeführt werden müssen.

5.1 Programmstruktur

Zuerst wird die Grundsteuerung des Robotinos über die serielle Schnittstelle in Verbindung der Webcam realisiert. Die Programmstruktur dazu wird übergeordnet in zwei Abschnitte geteilt. Ein Abschnitt wird direkt auf dem Robotino entwickelt und fungiert als Treiber für die Hardware und der andere Teil des Programms wird unter Matlab/Simulink und dem RTW Embedded Coder auf einem Windows Rechner konstruiert. Matlab/Simulink befasst sich dabei mit der Verarbeitung der Daten, die von den Treibern kommen oder an sie wieder gesendet werden sollen. Nach der Fertigstellung wird das Simulinkmodell in C-Codes gewandelt und mit den beschriebenen Kommunikationsprogrammen auf den Robotino übertragen. Die Verknüpfung dieser beiden Systeme findet anschließend über die ert_main.c Datei statt. Demnach wird zuerst das Simulinkmodell zur Verarbeitung der Daten erstellt und dann mit der Entwicklung der Treiber und deren Anknüpfung an das erstellte Modell fortgefahren. Die kompletten Quellcodes dazu sind im Anhang zu finden.

5.1.1 Simulinkmodell

Die Datenverarbeitung der Sensoren sowie der Bilder aus der Webcam und das setzen der Geschwindigkeitsgrößen des Robotinos wird in einem Simulinkmodell vorgenommen. Das Modell weist zu diesem Zweck einen Ausgangsport auf, der die Robotinogeschwindigkeiten und die Bumperinformation an die Treiber des Robotinos weiterleitet und elf Eingangsports, die einige Sensorinformationen und Graustufenbilder der Webcam in das Modell hineinführen. Mit den Embedded Matlab Funktionsblöcken können Programme zur Verarbeitung dieser Daten geschrieben und in die Anordnung eingebunden werden. Anhand der nachfolgende Abbildung 5.1 werden die einzelnen Komponenten dieses Modells erläutert:

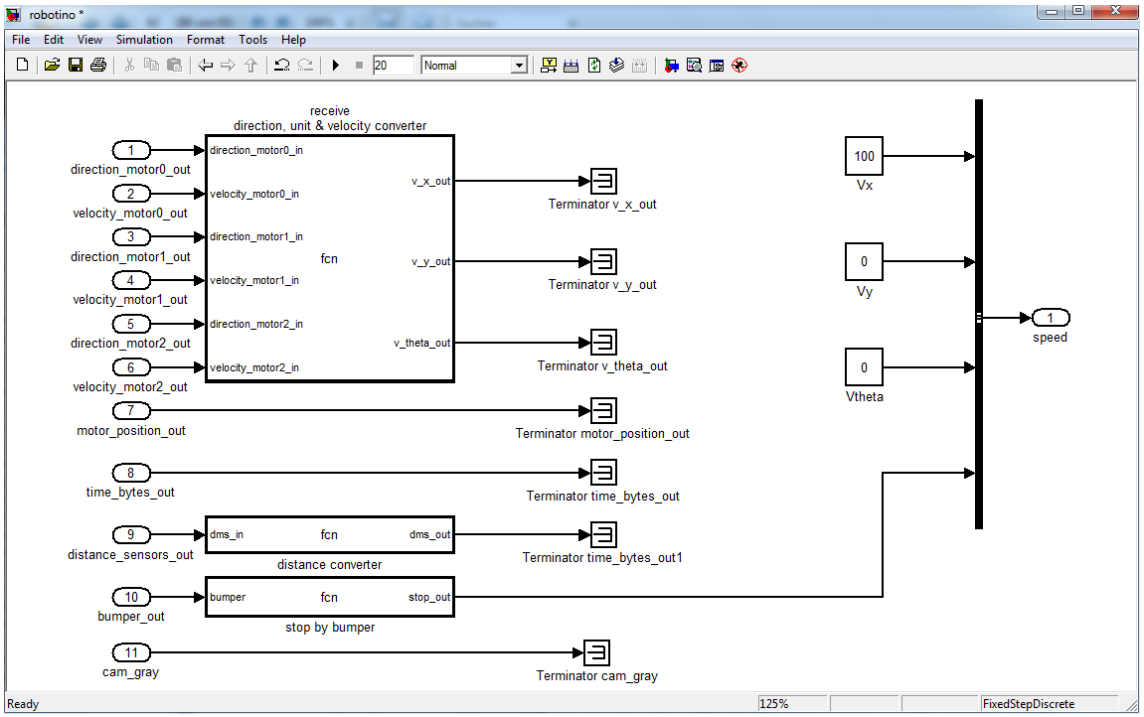

Abbildung 5.1: Simulinkmodell als Sensor- und Aktorschnittstelle

receive direction, unit & velocity converter: Dieser Embedded Matlab Block bekommt von den Robotino Treibern Informationen zu den momentanen Geschwindigkeiten jedes Antriebsmotors und deren Drehrichtung über die zugehörigen Eingangsports. Anhand dieser Daten werden in dem Block die drei Geschwindigkeitswerte Vx, Vy und Vtheta des Roboters in mm/s beziehungsweise rad/s berechnet und herausgegeben. Diese Informationen werden im weiteren Verlauf nicht verwendet und daher durch einen „*Terminator*“ abgeschlossen.

motor_position_out: Der Eingangsport 7 übermittelt über einen 12 Bytes langen Vektor die einzelnen Motorpositionen. Die Portdimension muss hierbei über die Portparameter auf 12 gesetzt werden. Jede Motorposition wird aus 4 Bytes zusammengesetzt. Die Motorpositionen werden im weiteren Verlauf nicht verwendet und daher durch einen „*Terminator*“ abgeschlossen.

time_bytes_out: Der Eingangsport 8 übermittelt über einen 9s Byte langen Vektor die einzelnen Motorlaufzeiten (Portdimension = 9). Jede Motorlaufzeit wird aus 3 Bytes zusammengesetzt. Die Motorlaufzeiten werden im weiteren Verlauf nicht verwendet und daher durch einen „Terminator“ abgeschlossen.

distance converter: In diesem Block werden die Rohdaten der neun (Portdimension des zugehörigen Eingangsports = 9) Infrarot Abstandssensoren in einem Abstandsbereich von 40 mm bis zu 300 mm liniarisiert. Werte außerhalb dieses Bereiches führen zu verfälschten Ergebnissen. Die Abstandssensoren werden bei der anstehenden Regelung nicht eingesetzt und daher durch einen „*Terminator*“ abgeschlossen.

stop by bumber: Der stop by bumper verarbeitet die Bumperaktivierung und setzt ein Signal an den Ausgangsport. Dieses Signal wird bei einer Kollision verwendet, um den Robotino zu stoppen und den Programmablauf zu beenden.

cam_gray: Dieser Eingangsport führt die aufgenommenen Bilder der Webcam im Graustufenformat in das Simulinkmodell hinein. Die Pixeldaten der Bilder werden in einem eindimensionalen Array zusammengesetzt. Die Portdimension muss folglich in Abhängigkeit der Bildauflösung der Webcam gewählt werden. Bei einer Auflösung von 160 x 120 Pixel muss beispielsweise diese auf 19200 gesetzt werden. Die Bilddaten werden hierbei noch nicht verwendet und daher durch einen „*Terminator*“ abgeschlossen.

Vx: In dieser Konstanten wird ein Wert für die Vorwärtsgeschwindigkeit gesetzt. Bei einem positiven Wert soll der Robotino nach vorne fahren und bei einem negativen Wert nach hinten. Hierbei beträgt dieser Wert 100 mm/s.

Vy: In dieser Konstanten wird ein Wert für die Seitwärtsgeschwindigkeit gesetzt. Bei einem positiven Wert soll der Robotino nach links fahren und bei einem negativen Wert nach rechts. Hierbei beträgt dieser Wert 0 mm/s.

Vtheta: In dieser Konstanten wird ein Wert für die Rotationsgeschwindigkeit gesetzt. Bei einem positiven Wert soll der Robotino gegen und bei einem negativen Wert mit dem Uhrzeigersinn drehen. Hierbei beträgt dieser Wert 0 rad/s.

Die drei Embedded Matlab Funktionsblöcke aus der Abbildung 5.1 wurden von dem Starterkitmodell von Herrn Dipl.-Ing. Elard Köhler übernommen. Diese Funktionen können aufgrund parallel aufgebauter Ansteuerungstechnik in diesem Fall simultan verwendet werden. Genauere Angaben zu der Berechnung der Robotinogeschwindigkeiten oder zur Liniarisierung der Abstandssensoren können in seiner Diplomarbeit „*Entwicklung einer kamerabasierten Bahnführungs-Regelung für einen omnidirektionalen Roboter mit Matlab xPC Target*“ nachgelesen werden.

5.1.2 Treiber zur seriellen Schnittstelle

Die Ansteuerung der drei Motoren des Robotinos und der Sensoren wird über die serielle Schnittstelle auf dem Robotino Betriebssystem durchgeführt. Die Termios Bibliothek bildet dabei die Grundlage für die Übertragung der vordefinierten Daten von Festo. Die nachfolgenden Bibliotheken

- init_ser.h
- drive.h

wurden dazu geschrieben und unter dem folgenden Pfad

```
/home/robotino/robotino_api/
```

auf dem Robotino abgelegt. Diese beiden Bibliotheken werden nach der Umwandlung des obigen Simulinkmodells (siehe Abbildung 5.1) und nach dem Kopieren der C-Codes auf

den Robotino durch den Header der ert_main.c Datei eingebunden und im Programmablauf aufgerufen. Mit der Funktion „*init_ser()*“ wird die serielle Schnittstelle für die Kommunikation mit dem I/O Board des Robotinos initialisiert. Die einzelnen Eigenschaften können aus dem Anhang entnommen und deren Bedeutung im Kapitel 4.3 EIA-232 Ansteuerung nachgelesen werden. Der Rückgabewert dieser Funktion ist ein Zeiger auf den geöffneten Port, der für die „*drive()* Funktion verwendet werden kann, um Motorgeschwindigkeiten und Sensordaten über die vordefinierten Transferblöcke (siehe Tabelle 3.3 und 3.4) an die Hardwarekomponenten zu senden oder zu empfangen. Der Rückgabewert der „*drive()*“ Funktion liefert dabei die Sensordaten, von denen nur ein bestimmter Teil in dem Simulinkmodell verwendet wird. Die Argumente der Funktion bestehen aus drei Geschwindigkeiten des Roboters, die Seitwärtsgeschwindigkeit, die Vorwärtsgeschwindigkeit und die Rotationsgeschwindigkeit, einer Information zur Bumperaktivierung und dem bereits erwähnten Zeiger auf die initialisierte serielle Schnittstelle. Bei einer Aktivierung des Bumpers werden alle Geschwindigkeiten auf null gesetzt und der Programmablauf abgebrochen. Auf diese Weise werden Schäden bei eventuellen Kollisionen verhindert.

5.1.2.1 Omnidirektionaler Antrieb

Der Robotino besteht aus drei, voneinander unabhängigen Antrieben, die den Roboter in jede beliebige Richtung fahren können und ihn dabei gleichzeitig um die eigene Achse drehen. Die Räder des Omnidirektionalen Antriebes sind um das Chassis in einem Winkel von 120° zueinander angebracht (siehe Abbildung 3.5). Jede Fahrrichtung muss aus den drei Motorgeschwindigkeiten zusammengesetzt werden. Diese Zusammensetzung der einzelnen Geschwindigkeiten wird in der „*drive()*“ Funktion vorgenommen.

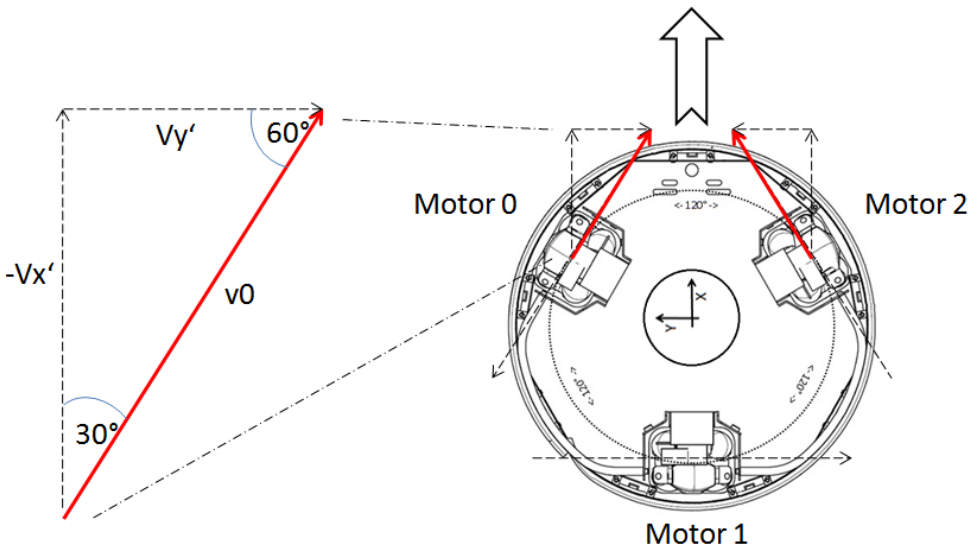

Abbildung 5.2: Berechnung der Vorwärtsfahrt (Robotino)

Die Motoren des Robotinos werden, wie in der Abbildung 5.2 dargestellt, gegen den Uhrzei-

gersinn nummeriert. Bei einer positiven Stellgröße einer Motorgeschwindigkeit drehen sich die Räder, von außen gesehen, im Uhrzeigersinn. Die Fahrrichtung des Robotinos wurde so definiert, dass für eine Vorwärtsbewegung ein positiver Vx Wert und für eine Seitwärtsbewegung, von dem Robotino aus gesehen nach links, ebenfalls ein positiver Vy Wert gesetzt werden muss. Bei einem positiven Rotationswert Vtheta dreht sich der Roboter um die eigene Achse, gegen den Uhrzeigersinn. Mit diesem Hintergrund wird die Motorgeschwindigkeit v0 am Beispiel der Abbildung 5.2 mit der nachfolgenden Formel berechnet:

$$v0 = -v_x \cdot cos(30^\circ) + v_y \cdot sin(30^\circ) + v_\theta \cdot 135mm \tag{1}$$

Aus der Formel (1) geht hervor, dass der Rotationswert Vtheta mit 135 mm multipliziert wird. Dieser Wert basiert auf dem Abstand eines Rades zum Mittelpunkt des Roboters (siehe Tabelle 3.1) und ergibt eine Winkelgeschwindigkeit des Roboters in rad/sec. Eine Kombination aller drei Motorgeschwindigkeiten und damit eine beliebige, resultierende Geschwindigkeit des Robotinos kann mit der Gleichung (1) und den nachfolgenden Gleichungen (2) und (3) berechnet werden:

$$v1 = \qquad -v_y + v_\theta \cdot 135mm \tag{2}$$

$$v2 = v_x \cdot cos(30^\circ) \qquad +v_y \cdot sin(30^\circ) + v_\theta \cdot 135mm \tag{3}$$

Anschließend werden die errechneten Motorgeschwindigkeiten mittels eines Faktors in mm/s umgerechnet. Dieser Faktor wurde durch Messfahrten von Herrn Dipl.-Ing. Elard Köhler in seiner Diplomarbeit „*Entwicklung einer kamera-basierten Bahnführungs-Regelung für einen omnidirektionalen Roboter mit Matlab xPC Target*“ bestimmt und kann aufgrund identischer Ansteuerungstechnik auch hierbei verwendet werden. Für die Umrechnung ergibt sich damit der nachfolgende Zusammenhang:

$$v0 = \frac{v0}{7,02} \tag{4}$$

$$v1 = \frac{v1}{7,02} \tag{5}$$

$$v2 = \frac{v2}{7,02} \tag{6}$$

5.1.2.2 Fehler beim Lesen der Sensordaten

Bei der Größe der Sensordaten können einzelne Bytes während des Programmablaufs verloren gehen. Diese Verluste können anhand der drei Start- und Stoppbytes des Empfangsblocks erkannt werden (siehe Tabelle 3.4). Falls ein gelesener Empfangsblock nicht diese Zeichen auf den vorgeschriebenen Position enthält, wird davon ausgegangen, dass die gelesenen Daten fehlerhaft sind und infolgedessen werden diese nicht bei der Datenauswertung verwendet. Diese Abfrage wurde mit einer „*if*“ Anweisung in der ert_main.c Datei realisiert.

5.1.3 Treiber zur Webcam

Die Ansteuerung der USB Webcam wurde ebenfalls durch einen eigen erstellten Treiber realisiert. Video4linux und video4linux2 bilden dabei die Grundlage für die Aufnahme der Bilder mit den passenden Eigenschaften. Dementsprechend wurde die Bibliothek

- grab.h

geschrieben und zusammen mit „*init_ser.h*“ und „*drive.h*“ in das Robotino Betriebssystemverzeichnis

```
/home/robotino/robotino_api/
```

abgelegt. Diese Bibliothek lässt sich genau so wie die Vorgänger, nach dem Kopieren der erzeugten C-Codes aus dem Simulinkmodell auf den Robotino durch den Header der ert_main.c Datei einbinden und im Programmablauf aufrufen. Der Aufruf der Funktion „*init_webcam()*“ mit den zugehörigen Funktionsargumenten öffnet einen Kommunikationskanal zur Webcam und initialisiert die Aufnahme der Kamera mit individuellen Einstellungen. Zu den Argumenten des Aufrufs gehört einmal der Zeiger auf die geöffnete Webcam und eine Struktur, die die Videoinformationen beinhaltet. Nach dem die Webcam initialisiert worden ist, können mit dem Funktionsaufruf „*take_picture()*“, im späteren Programmverlauf, Grauwertbilder aus dem Videostream der Kamera gemacht werden. Hierfür wird für die Argumente dieser Funktion der Pointer auf die geöffnete Webcam eingesetzt, die Pixelhöhe und Pixelbreite des Bildes und ein vierdimensionales Array für die Pixeldaten des Bildes. Bei den ersten drei Dimensionen handelt es sich um die drei Grundfarben rot, grün und blau, die zusammen ein Farbbild ergeben. Die vierte Dimension beinhaltet das Graustufenbild, das aus dem RGB Farbraum berechnet wird. Die Umrechnung des Farbbildes zum Graustufenbild wird nach der gleichen Formel gemacht, wie aus der Image Processing Toolbox von Matlab. Dazu wird die nachfolgende Gleichung verwendet:

$$Grau = R \cdot 0,2989 + G \cdot 0,5870 + B \cdot 0,1140 \tag{7}$$

Die Berechnung aus der Gleichung (7) wird für jedes Pixel des gesamten Bildes vorgenommen. Eine weitere optionale Funktion „*picture_to_file*“ aus der eigen erstellten grab.h Bibliothek kann benutzt werden, um das aufgenommene Bild auf dem Robotino abzuspeichern. Dieses Bild kann nachträglich auf den Windows Rechner kopiert und mit der Matlab Image Processing Toolbox untersucht werden.

5.1.4 Zusammensetzung der Programmabschnitte

Die beiden Programmabschnitte werden sinngemäß auf dem Robotino zusammengesetzt. Dafür wird zu Beginn das Simulinkmodell aus der Abbildung 5.1 mit den RTW Embedded Coder und der konfigurierten tmf-Datei in C-Codes gewandelt und über die ert_main.c Datei die Treiber der seriellen Schnittstelle und der USB Webcam angehängt. Dabei werden

an die jeweiligen Eingangs- und Ausgangsports aus dem Modell die zugehörigen Treiberinformationen übergeben. Danach kann der komplette Ordner „*robotino_ert_rtw*“ in das Robotinoverzeichnis

```
/home/robotino/
```

kopiert und von dort aus kompiliert und ausgeführt werden. Bei diesem Aufbau wird der Robotino mit einer Geschwindigkeit von 100 mm/s vorwärts fahren, bis das Programm nicht manuell abgebrochen wird oder bis der Bumper nicht aktiviert wird.

5.2 Prozessautomatisierung

Die Programmzusammensetzung und der anschließende Start der Robotinofahrt beinhalten mehrere Schritte. Zu diesen gehört die nachfolgende Abfolge:

1. Umwandlung des Simulinkmodells in C-Codes
2. Überschreiben der ert_main.c Datei
3. Kopieren der C-Codes auf den Robotino
4. Kompilieren der C-Codes auf dem Robotino
5. Start der Robotinofahrt
6. Runterladen der Simulationsdatei vom Robotino

Diese Schrittabfolge kann durch einige Skriptdateien auf dem Windows Rechner automatisiert werden, sodass nur einzelne Mausklicks diese Prozesse durchführen. Nachfolgend wird ein Beispiel gezeigt, wie diese Skriptdateien aufgebaut sind und wie sie benutzt werden.

5.2.1 Windows-Batchdateien

Die Windows-Batchdateien sind Skriptdateien für die DOS Eingabeaufforderungen, in denen DOS Befehle zusammengefasst und durch deren Aufruf nacheinander abgearbeitet werden. Diese Datei kann in diesem Fall eingesetzt werden, um das Überschreiben der ert_main.c zu automatisieren oder um WinSCP mit einem zugehörigen Skript, das die erstellten C-Codes aus dem Robotino Simulinkmodell auf den Robotino kopiert, auszuführen. Die Batchdatei kann mit Hilfe eines beliebigen Texteditors erstellt und bearbeitet werden. Damit sie als solche auch von dem System erkannt werden kann, muss sie mit der Endung „*.bat*“ abgespeichert werden. Für die, in diesem Abschnitt genannten Aufgaben, wird „*copy.bat*“ eingesetzt und im selben Verzeichnis wie „*robotino.mdl*“ wie folgt aufgebaut

```
@echo off
color f1
xcopy ert_main.c %cd%\robotino_ert_rtw
„winscp.exe Pfad“ /console /script=“WinSCP Skriptpfad“
echo Kopiervorgang beendet!
exit
```

Beim Aufruf der obigen Batchdatei wird in den ersten beiden Zeilen die Ausgabe des nachfolgenden Befehls unterdrückt und der Hintergrund der Eingabeaufforderung auf weiß gesetzt. Die nächste Zeile kopiert die ert_main.c Datei, die sich in dem gleichen Ordner wie die Batchdatei befinden sollte, in das von dem RTW Embedded Coder erzeugte „*robotino_ert_rtw*“ Verzeichnis auf dem Windows Rechner. Für WinSCP können ebenfalls spezielle Skripte erzeugt werden, deren Einsatzmöglichkeiten auf der WinSCP Homepage nachgelesen werden können. In der vierten Zeile wird WinSCP mit so einem zugehörigen Skript gestartet. Hierbei müssen die Pfade von „*winscp.exe*“ und der „*copy.bat*“ eingesetzt werden. Dieser Aufruf baut eine Verbindung über WLAN mit dem Robotino auf, kopiert danach den „*robotino_ert_rtw*“ Ordner in das vorgesehene Verzeichnis auf dem Robotino und kompiliert anschließend die C-Codes. Zum Abschluss wird eine Signalinformation ausgegeben und die Eingabeaufforderung geschlossen. Nach diesem Prinzip kann auch die Simulationsdatei vom Robotino mit „*sim.bat*“ heruntergeladen werden. Der Inhalt dieser und der zugehörige WinSCP Skript kann aus dem Anhang entnommen werden.

5.2.2 WinSCP Skript

Das WinSCP Script ist eine gängige „*.txt*“ Datei, in der spezielle Befehle für die Kommunikation über WinSCP zusammengefasst und der Reihe nach ausgeführt werden können. Die Einsatzmöglichkeiten dafür können auf der jeweiligen Homepage nachgelesen werden. Für den Datenaustausch mit dem Robotino wurde „*copy.txt*“ mit dem folgenden Inhalt erstellt

```
option batch off
option confirm on
open robotino:robotino@192.168.0.13
cd /home/robotino
rm robotino
rm simout.csv
rmdir robotino_ert_rtw
option transfer binary
put robotino_ert_rtw
cd /home/robotino/robotino_ert_rtw
call make -f robotino.mk
exit
```

Dieses Skript wird von „*copy.bat*“ aufgerufen, um das Verzeichnis „*robotino_ert_rtw*“ vom Windows Rechner auf den Robotino zu übertragen und die beinhaltenden C-Codes zu kompilieren. Zu diesem Zweck werden in den ersten beiden Zeilen Optionen gesetzt, die die Befehlsausgabe in der Eingabeaufforderung unterdrücken und eine manuelle Kopierbestätigung aktivieren. Danach wird ein Kommunikationskanal über die Robotino IP mit den zugehörigen Logindaten aufgebaut. Dafür muss eine WLAN Verbindung zum Access Point vorhanden sein. In den Zeilen vier bis neuen wird in den Robotino Ordner auf dem Linux Betriebssystem navigiert, dort die alten Dateien gelöscht und die neuen C-Codes vom Windows Rechner kopiert. Anschließend wird das übertragene Verzeichnis geöffnet, der „*make*“ Befehl aufgerufen und das Programm beendet.

5.2.3 Aufruf der Skripte unter Simulink

Die Windows-Batchdateien mit den zugehörigen WinSCP Skripten können direkt aus der Eingabeaufforderung gestartet werden. Matlab bietet noch eine weitere Möglichkeit diese Aufrufe direkt aus dem Simulinkmodell durchzuführen. Dafür müssen alle Skriptdateien sowie die zu übertragenen Daten im selben Verzeichnis wie „*robotino.mdl*“ abgelegt werden. Hierzu wird in dem Robotino Simulinkmodell ein beliebiger Text geschrieben, beispielsweise „*Build/Compile Model*“ und über die Rechte Maustaste darauf „*Annotation properties...*“ aufgerufen. In dem geöffneten Fenster werden die Befehle unter „*ClickFcn*“ eingetragen, die beim Klicken auf den ausgewählten Text ausgeführt werden. An dieser Stelle kann auch der Aufruf der Batchdatei eingefügt werden. Die Nachfolgende Abbildung 5.3 verdeutlicht nochmal diesen Zusammenhang:

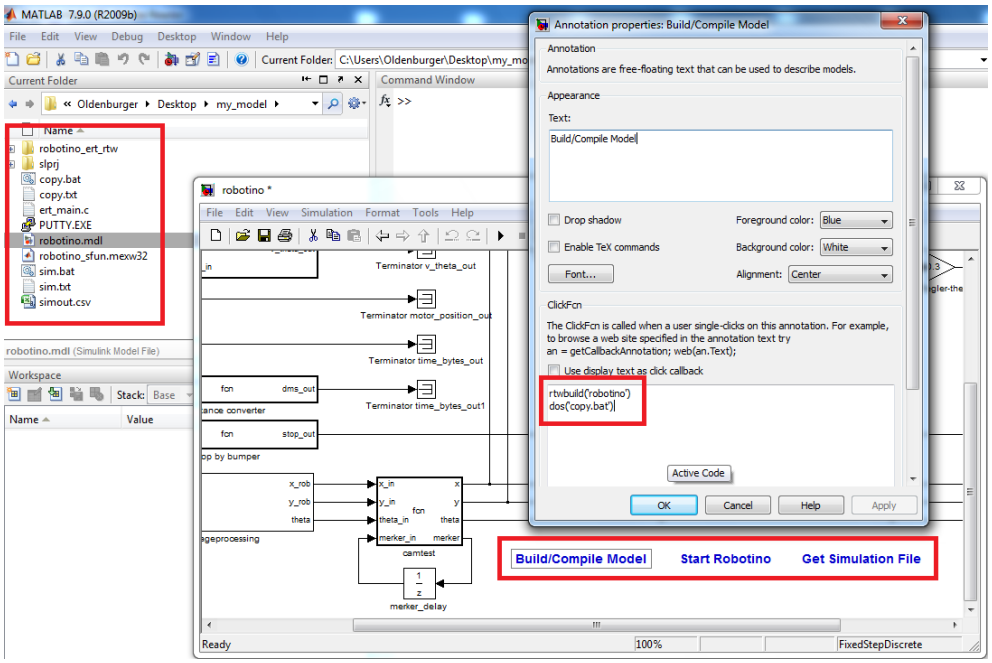

Abbildung 5.3: Flickfunktion für Skriptaufrufe

Nach gleichem Vorgehen wurde auch die Klickfunktion zum Runterladen der Simulationsdatei von dem Robotino in das aktuelle Verzeichnis erstellt.

5.2.4 Start Robotino

Das Starten und Stoppen der Robotinofahrt wurde nach einer anderen Methode realisiert. Hierzu wird das Programm „*putty.exe*“ in das Robotino Verzeichnis eingefügt und über die Klickfunktion aus dem Simulinkmodell mit dem Befehl „*dos('putty.exe robotino@192.168.0.13 -pw robotino')*“ gestartet (siehe Abbildung 5.3). Mit diesem Aufruf meldet sich Putty über die Logindaten am Robotino an und öffnet ein Terminal unter Windows, in dem die Linuxbefehle eingegeben und auf dem Robotino ausgeführt werden können. Der Robotino wird mit der Eingabe

```
sudo ./robotino
```

gestartet und über die Tastenkombination „*STRG + C*“ wieder gestoppt. Dies hat den Verteil, dass die Fehlermeldungen durch das geöffnete Puttyfenster ausgegeben werden und der Robotino zur jeder Zeit angehalten werden kann. Ein Stoppen des Robotinos über WinSCP ist nicht möglich, da die Tastenkombination zum Abbruch der Prozesse nicht unterstützt wird.

5.3 Signalintervall

Der Antrieb des Robotinos ist aus Sicherheitsgründen so ausgelegt, dass die Motordrehung in Schritten durchgeführt wird. Diese Motordrehung muss für eine ruckelfreie Fahrt, ein kontinuierliches Signal erhalten, das die Bewegung der Räder aufrecht erhält. Ein Ausbleiben des Signals würde zum Stoppen des Roboters führen und bei einem zu schnellen Empfangsintervall könnten die Befehle nicht korrekt verarbeitet werden. Aus diesem Hintergrund wurde ein Zeitabhängiger Programmablauf entwickelt, bei dem die Schrittzeit in einem bestimmten Intervall eingehalten wird. Dieses Verhalten kann auch als weiche Echtzeit bezeichnet werden. Bei dieser Art der Echtzeit kann eine durchgeführte Aktion gelegentlich über den gesetzten Zeitrahmen hinausgehen, ohne das dabei weiterführende Folgen für die Steuerung des Robotinos bleiben.

5.3.1 Realisierung der Echtzeit

Der RTW Embedded Coder führt das konstruierte Programm als „*Single Tasking*“ aus. Dies bedeutet, dass bei diesem Echtzeitverhalten alle Funktionen solange hintereinander abgearbeitet werden, bis jede Aktion nicht vollständig durchgeführt worden ist. Die Aktionszeit ist abhängig von dem Rechenaufwand jedes Tasks und schwankt dementsprechend immer um einen bestimmten Wert. Damit die Rechenzeit eines Zyklus des Robotino Simulinkmodells gemessen werden kann, wurden in die ert_main.c Datei zwei weitere Bibliotheken

- sys/time.h
- time.h

eingebunden, die beim Start des Programmablaufs und am Ende eines Schleifenzyklus die Systemzeit aufnehmen können. Die erste Bibliothek benutzt dafür die nachfolgende Struktur

```
1 struct timeval {
2     tv_sec
3     tv_usec
4 }
```

Diese beinhaltet unter „*tv_sec*“ die momentane Systemzeit in Sekunden und unter „*tv_usec*“ in Mikrosekunden. Die Mikrosekunden werden immer bis zur vollen Sekunde hochgezählt und dann wieder *resettet*[50]. Wegen dem Resetten der Mikrosekunden müssen für den kompletten Zeitablauf des Robotinos die Sekunden dazu gezählt werden. Diese Zählung übernimmt die zweite Zeitbibliothek, in deren Struktur „*time_t*“ die Sekunden des Systems abgelegt werden. Um daraus eine weiche Echtzeit zu realisieren, wird ein fester Zeitrahmen in der ert_main.c Datei gewählt, in dem beinahe jeder Zyklus abgearbeitet werden kann. Die Differenz zwischen der aktuellen Zykluszeit und dem festgelegten Zeitrahmen wird mit einer Wartefunktion gefüllt. Dadurch wird eine feste Laufzeit erstellt, die hierbei als weiche Echtzeit bezeichnet werden kann. Das gelegentliche Überschreiten der festgelegten Zeitrahmens wird mit einer Fehlermeldung im Terminal angezeigt und der Programmablauf mit einer verspäteten Schleifenzeit fortgesetzt.

5.3.2 Kommunikationstest

Die konstruierte Echtzeit ermöglicht eine Messung des Signalintervalls, in dem der Robotino die Fahrbewegungen ohne zu Ruckeln ausführen kann. Diese Zeiten müssen bei den Regelabläufen bezüglich der Bildauswertung eingehalten werden. Für den Test der Kommunikation wird das Simulinkmodell aus der Abbildung 5.1 verwendet. Aufgrund des Eingangs mit den Graustufenbildern, ist eine minimale Signalintervallmessung nicht nötig, da in jedem Fall der Rechenaufwand für die Bilderfassung diesen Zeitpunkt überschreitet. Bei der kleinsten Auflösung der Kamera braucht das Programm ca. „*30 ms*“, um einen Schleifenzyklus durchzuführen. Für die Ermittlung des maximalen Signalintervalls wird der Zeitrahmen der Echtzeit bei einer Vorwärtsfahrt des Robotinos langsam erhöht, bis der Roboter visuell zu ruckeln beginnt. Dies geschieht bei einem Schleifenzyklus der länger als „*0,85 Sekunden*“ dauert.

[50] Zurücksetzen der Zeit

5.4 Entwicklung des Bezugspunkts

Für eine Koordination des Robotinos in einem Raum wird ein Bezugspunkt entwickelt, der über die USB Webcam fehlerlos identifiziert werden kann. Anhand dieser Identifizierung ist es dann möglich, über den Pixelabstand auf dem Kamerabild eine umgerechnete Position des Roboters zum Bezugspunkt zu berechnen. Für Integrationszwecke des Systems wird zuerst ein einfaches Merkmal verwendet, das über den Robotino an der Decke angebracht wird. Dieses Merkmal muss sich deutlich von dem Deckenhintergrund unterscheiden. Demnach werden helle LED's verwendet, die von der Kamera als hellster Punkt in der Umgebung wahrgenommen werden. Dieser Punkt bildet den definierten Koordinatenursprung der Robotinoposition. In der nachfolgenden Abbildung 5.4 ist eine konzeptionelle Illustration zu diesem Hintergrund dargestellt:

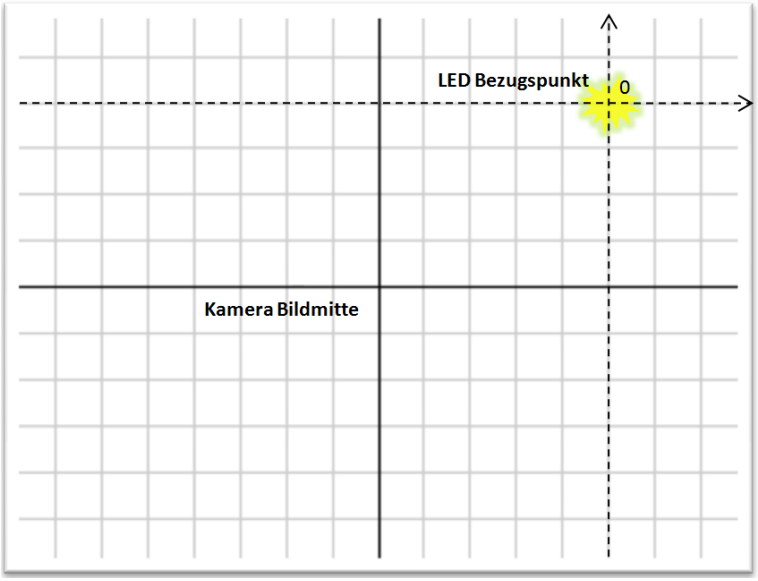

Abbildung 5.4: Konzept des Bezugspunkts

5.4.1 Weiterentwicklung des Bezugspunkts

Anhand des Bezugspunkts mit dem einfachen Merkmal kann nur die Position des Robotinos im Raum berechnet werden. Für weiterführende Regelabläufe wird ein weiteres Muster entwickelt, mit dem auch die Ausrichtung errechnet werden kann. Hierfür werden drei LED's verwendet, die als Muster ein gleichschenkliges Dreieck auf der Decke über den Robotino bilden. Bei diesem Muster ist die Dreieckspitze in etwa doppelt so weit entfernt von den anderen Ecken, wie diese zu einander. Angesichts dieser LED Konstruktion kann ein Winkel im Bezug auf die X-Achse des Kamerabildes bestimmt werden, der mit der Ausrichtung des Robotinos verknüpft werden kann. Die Mitte des Dreiecks wird als Koordinatenursprung genommen für die Robotinoposition. In der nachfolgenden Abbildung 5.5 ist eine konzeptionelle Illustration zu diesem Hintergrund dargestellt:

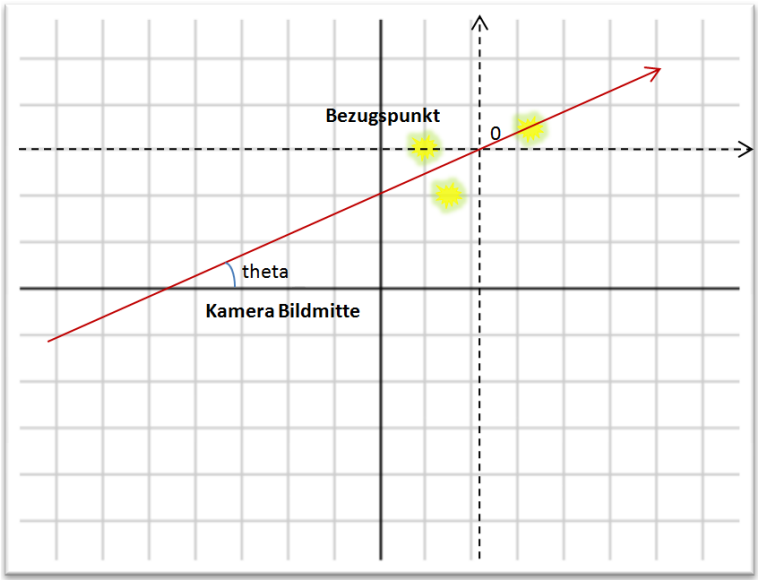

Abbildung 5.5: Erweitertes Konzept des Bezugspunkts

5.5 Webcamjustierung

Für die USB Webcam werden einige Voreinstellungen und Tests durchgeführt, mit denen die Erkennung des Bezugspunkts auf der Decke bei den Regelfahrten fehlerfreier funktioniert. Für diese Erkennung wird überprüft, welche Kameraauflösungen möglich sind, die eine ruckelfreie Bewegung des Robotinos erlauben. Die Auflösung spielt ebenfalls eine große Rolle bei dem Rechenaufwand der digitalen Bildverarbeitung und muss dementsprechend bei diesen Test mitberücksichtigt werden. Durch die Umwandlung der Bildpixel in cm wird die Genauigkeit der Koordinationsberechnung bestimmt. Für einen besseren Kontrast zwischen dem Bezugspunkt und dessen Deckenhintergrund können einige Einstellungen an der Kamera mit Hilfe der vorhandenen Bibliotheken gemacht werden. An dieser Stelle wird auch untersucht, ob zusätzliche Komponenten an der Kamera den Wirkungsgrad der Mustererkennung erhöhen können.

5.5.1 Zeitmessung bei verschiedenen Auflösungen

Für die Zeitmessung der Webcam bei verschiedenen Auflösungen wird ein neues Programm „*camspeed*“ geschrieben. In diesem Programm werden die gleichen Bibliotheken eingesetzt, wie bei der Steuerung des Robotinos. Die Messungen der Bildraten bei unterschiedlichen Auflösungen ergeben somit die nachfolgenden Werte:

Auflösung	Bildrate
160 x 120	31ms (ca. 30 Bilder/Sek)
176 x 144	31ms (ca. 30 Bilder/Sek)
320 x 240	65ms (ca. 15 Bilder/Sek)

352 x 288	65ms (ca. 15 Bilder/Sek)
640 x 480	65ms (ca. 15 Bilder/Sek)

Tabelle 5.1: Gemessene Bildraten der Logitech E3500 Webcam

Im Vergleich der obigen Tabelle 5.1 mit den gemessenen Werten und der Tabelle 3.8 mit den Angaben des Herstellers ist zu erkennen, dass bei den höheren Auflösungen es einen deutlichen Unterschied in der Bildrate gibt.

5.5.1.1 Rechenaufwand der digitalen Bildverarbeitung

Die Messungen der Bildraten bei unterschiedlichen Auflösungen zeigen, dass jeder dieser Werte das geforderte Signalintervall des Robotinos für eine ruckelfreie Fahrt einhält. Bei einer digitalen Bildverarbeitung werden alle Pixel eines Bildes nach bestimmte Kriterien mindestens einmal bei jeder Bildoperation untersucht. Bei der kleinsten Auflösung eines Grauwertbildes sind es insgesamt 19200 Bildpixel und bei der höchsten Auflösung 307200 Bildpixel, die analysiert werden müssen. Im Vergleich ist eine Bildoperation mit der höchsten Auflösung 16 mal Rechenaufwendiger, als eine mit der kleinsten Auflösung und aus diesem Grund muss der Zeitaufwand der anstehenden Bildverarbeitung mit berücksichtigt werden. Das Programm „*camspeed*“ wird zweckmäßig so modifiziert, dass eine Operation der digitalen Bildverarbeitung eingebaut und bei verschiedenen Auflösungen dessen Zeitverhalten gemessen wird. Bei dieser Bildoperation handelt es sich um eine einfache Binarisierung des Bildes. Dabei wird eine Graustufengrenze gewählt und alle unterliegenden Grauwerte auf 0 (schwarz) und alle darüber liegenden Werte auf 1 (weiß) gesetzt. Zum Vergleich wurden diese Messungen auch auf einem leistungsstärkeren Rechner durchgeführt. Es wurden die nachfolgenden Zeiten gemessen:

Auflösung	**Robotino**	**Rechner**
160 x 120	31ms (ca. 30 Bilder/Sek)	31ms (ca. 30 Bilder/Sek)
176 x 144	31ms (ca. 30 Bilder/Sek)	31ms (ca. 30 Bilder/Sek)
320 x 240	93ms (ca. 11 Bilder/Sek)	65ms (ca. 15 Bilder/Sek)
352 x 288	130ms (ca. 8 Bilder/Sek)	65ms (ca. 15 Bilder/Sek)
640 x 480	374ms (ca. 3 Bilder/Sek)	65ms (ca. 15 Bilder/Sek)

Tabelle 5.2: Zeitmessung der Bildaufnahme mit Binarisierung

Bei einer digitalen Bildverarbeitung zur Erkennung des weiterentwickelten Bezugspunkts werden ca. 10 Operationen angesetzt. Aus der obigen Tabelle 5.2 geht hervor, dass bei diesem Vorhaben der Einsatz der höchsten Auflösung das Signalintervall für die Steuerung des Robotinos weit überschreiten wird. Die mittlere Auflösung (320 x 240) könnte unter umständen gerade noch den Maximalwert des Signalintervalls einhalten, jedoch wäre eine flüssige Regelung dabei nur schwer realisierbar. Angesichts dieser Voraussetzungen wird bei der vorhandenen Hardware mit der minimalen Bildauflösung gearbeitet. Beim Vergleich

der Messzeiten von einem leistungsstärkerem Rechner mit der Tabelle 5.1 fällt auf, dass die digitale Bildverarbeitung bei diesem Prozess keinen deutlichen Zeitaufwand benötigt. Demzufolge wäre mit einem neueren Rechner auf dem Robotino eine Bildverarbeitung mit der höchsten Auflösung durchaus denkbar.

5.5.2 Pixelumrechnung

Die Pixelumrechnung der Kamerabilder in eine reale Länge in cm wird anhand einer Aufnahme einer Messmatte durchgeführt. Die Messmatte ist genau 1,4 m lang, 1 m breit und die Fläche ist in 1 dm^2 große Quadrate aufgeteilt. Diese Matte wird auf dem Boden gelegt und im gleichen Abstand, wie die Robotino Webcam zum Bezugspunkt, eine Aufnahme mit einer Bildauflösung von 160 x 120 gemacht. Mit der Image Processing Toolbox von Matlab wird im aufgenommenen Bild die Pixelanzahl der Matte in der Länge und Breite gemessen (siehe Abbildung 5.6).

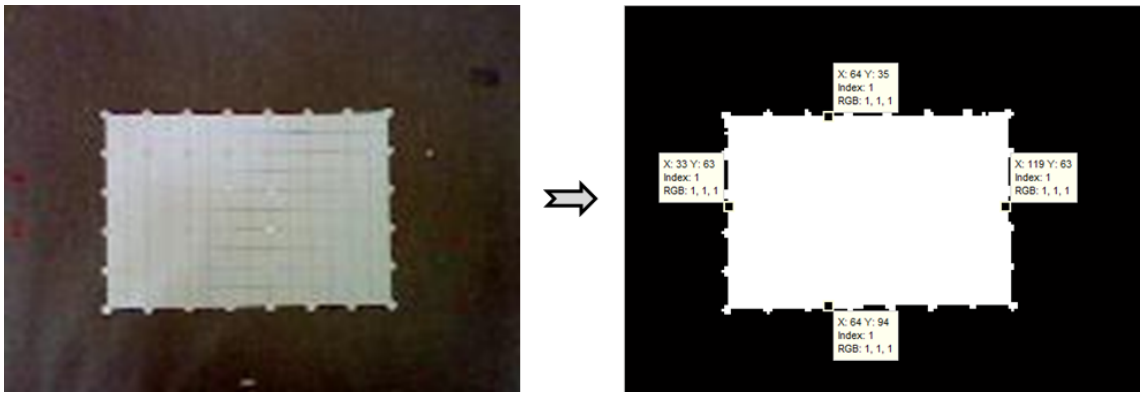

Abbildung 5.6: Pixelmaße der Messmatte

Dazu wurde das aufgenommene Bild binarisiert und die Grenzen der Matte im Bild mit einem Matlab Bildwerkzeug markiert. Aus diesen Markierungen ergeben sich folgende Werte für die Pixel in cm Umrechnung:

$$Laenge = \frac{140cm}{(94-35)Pixel} = 1,6949\frac{cm}{Pixel} \quad (8)$$

$$Breite = \frac{100cm}{(119-33)Pixel} = 1,6279\frac{cm}{Pixel} \quad (9)$$

Nach der Bildung des Mittelwerts aus den Ergebnissen der Gleichungen (8) und (9) ergibt sich für die Pixelumrechnung in cm:

$$1Pixel = 1,66cm \quad (10)$$

Dieser Umrechnungsfaktor sorgt für eine maximale, 1,66 cm genaue Bestimmung des Robotinos auf dem Boden, mittels der Webcam und des Bezugspunkts. Wegen schwankender

Bildqualität durch Änderungen des Tageslichts, ist die tatsächliche Berechnung der Koordinaten noch etwas ungenauer.

5.5.3 Kameraeinstellungen

Die Einstellungen der Kamera können empirisch über die Webcambibliotheken geändert werden, um einen ausgeprägteren Unterschied zwischen dem Bezugspunkt und dem Hintergrund in der Bildaufnahme zu erhalten. Zu diesem Zweck werden von dem Bezugspunkt Aufnahmen mit der Robotino Webcam gemacht, abgespeichert und visuell beurteilt. Die Kamera wird dabei an dem Robotino befestigt und in die Fahrtrichtung ausgerichtet. In erster Linie wird die automatische Belichtungs- und Lichtverstärkungsregelung abgeschaltet und dann mit der Kombinierung mehrerer Einstellungen fortgefahren. Für die Bildaufnahmen werden die beschriebenen Funktionen aus dem Webcamtreiber verwendet. Der deutlichste Unterschied ergibt sich bei diesem Vorgehen bei den folgenden Konfigurationen:

- **V4L2_CID_EXPOSURE_AUTO** = 0
- **V4L2_CID_GAIN** = 255
- **V4L2_CID_EXPOSURE_ABSOLUTE** = 1000
- **brightness** = 2000
- **contrast** = 65535

Durch einen hohen Kontrast, einer starken Lichtverstärkung und einer relativ geringen Belichtung wird das Muster auf der Decke beinahe mit einem Grauwert von 255 (weiß) im aufgenommenen Bild belegt und der Hintergrund, bis auf selten entstehende Lichtreflektionen, mit dem Grauwert 0 (schwarz). Der Unterschied zu den Bildaufnahmen vor den empirischen Einstellungen und danach wird in der nachfolgenden Abbildung 5.7 verdeutlicht:

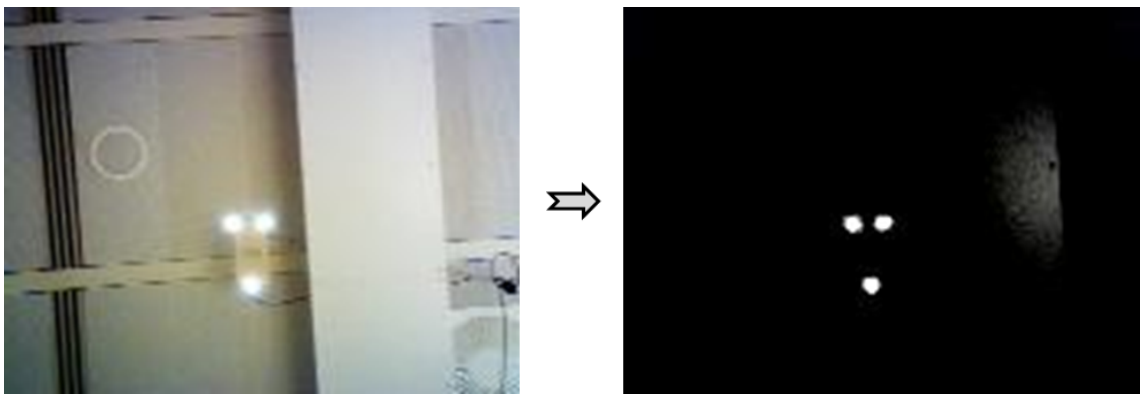

Abbildung 5.7: Vergleich der Kameraaufnahmen

Nach diesen Einstellungen kann der Bezugspunkt viel besser mit der digitalen Bildverarbeitung identifiziert werden. Die entstandene Lichtreflektion aus dem Vergleich der Kameraaufnahmen in der Abbildung 5.7 fällt dabei nicht ins Gewicht, da für die Bezugspunkterkennung ein viel höherer Schwellenwert für die relevanten Graustufen gesetzt und die Reflektion dadurch entfernt wird.

5.5.4 Zusätzliche Komponenten

Die enorme Helligkeit der LED's von dem Bezugspunkt führt zu einer Überbelichtung der zugehörigen Bereiche auf dem Kamerasensor. Um dieser Überbelichtung entgegenzuwirken, wird eine zusätzliche Komponente an die Kamera angebracht, die die Lichtintensität dämpfen und die Bezugspunkterkennung verbessern soll. Dazu dient eine abgedunkelte Plastikfolie, die auf die Linse der Webcam angebracht wird (siehe Abbildung 5.8).

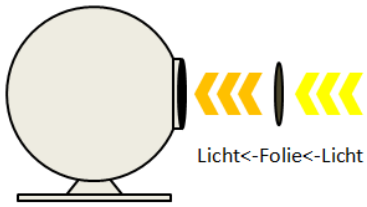

Abbildung 5.8: Folie zur Dämpfung der Lichtintensität

Mit dieser Komponente an der Kamera wird der Bezugspunkt aufgenommen und visuell beurteilt. Für eine deutlichere Erkennung des Wirkungsgrades wird die Aufnahme ohne die vorher empirisch ermittelten Einstellungen der Kamera durchgeführt und in der nachfolgenden Abbildung 5.9 dargestellt:

Abbildung 5.9: Bezugspunktaufnahme mit abdunkelnder Folie

In der obigen Abbildung ist zu erkennen, dass die Lichtintensität etwas geringer ist und die Aufnahme insgesamt dunkler wirkt. Das Licht des Musters wurde jedoch durch die

Folie zerstreut, sodass der Wirkungsgrad der Erkennung der einzelnen LED's ehe geringer ausfällt als vorher. Diese Nachteile machen den Einsatz dieser Folie an der Kamera für die Zielsetzung nicht vorteilhaft und somit wird diese im weiteren Verlauf nicht verwendet.

5.6 Einfache Punktregelung

Die einfache Punktregelung wird mit Hilfe des Bezugspunkts aus der Abbildung 5.4 hinsichtlich eines Integrationstests der Regelverfahren realisiert und die zugehörigen Programmelemente erläutert. Im Hinblick darauf wird zuerst eine Simulation in Matlab/Simulink entwickelt, die das Verhalten des Roboters beschreibt. Für den anschließenden Vergleich der Simulation und der realen Fahrt müssen zeitabhängige Positionsdaten des Robotinos während der Fahrt aufgezeichnet werden. Für die Aufzeichnung der Daten wird während der Fahrt eine Simulationsdatei mit den aktuellen Daten geschrieben, die von Matlab gelesen und anhand von Plots untersucht werden kann.

5.6.1 Mathematische Beschreibung des Robotinos

Der Robotino wird mathematisch in zwei unterschiedlichen *Subsystemen*[51] beschrieben, die in einem Simulinkblock „*Robotino*“ zusammengefasst werden. Dabei wird zwischen den Vorberechnungen des EIA-232 Treibers und der eigentlichen Hardwareumsetzung unterschieden (siehe Abbildung 5.10).

[51] Unterfunktionen von Simulinkblöcken

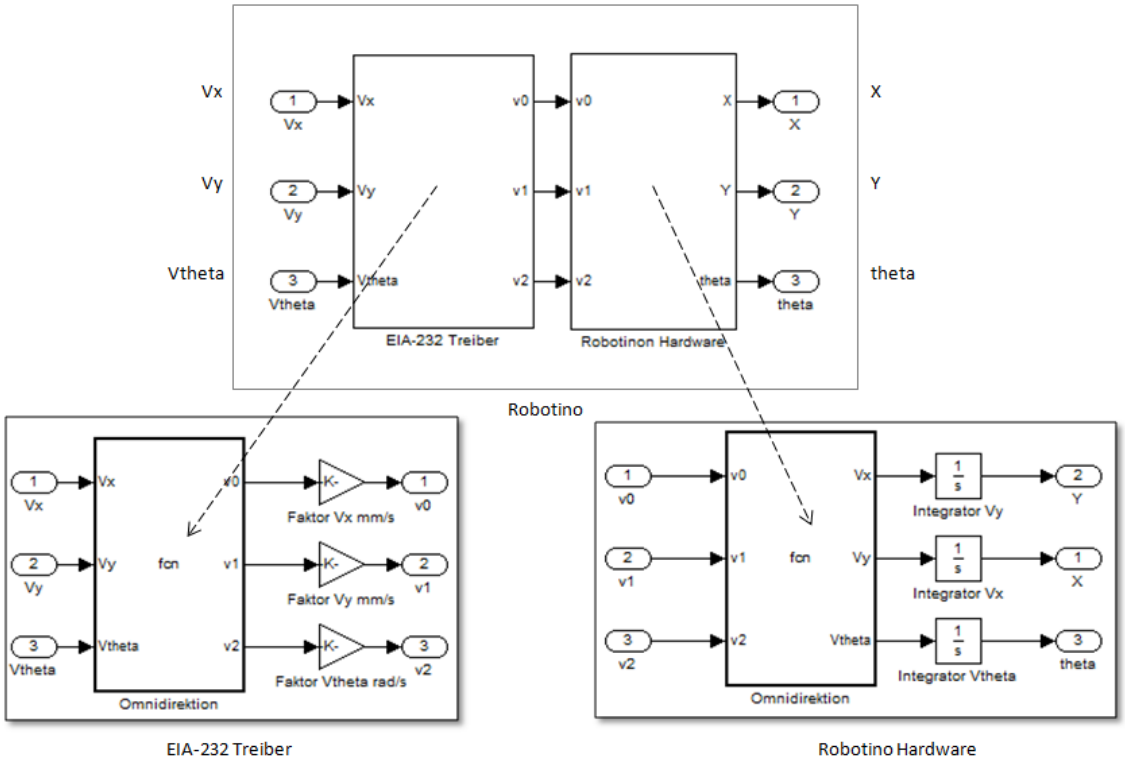

Abbildung 5.10: Simulinkmodell des Robotinos

Diese mathematische Beschreibung des Robotinos wird auch in den nachfolgenden Simulationsmodellen gleichwertig verwendet, um das entsprechende Regelverhalten des Roboters vor der realen Fahrt zu analysieren.

5.6.1.1 EIA-232 Treiber

In dem eigen erstellten Treiber der seriellen Schnittstelle werden die gesetzten Geschwindigkeiten Vx, Vy, und Vtheta, die von dem Simulinkmodell kommen, in einzelne Motorgeschwindigkeiten umgerechnet und diese dann nach der Wandlung mit den Umrechnungsfaktoren (4), (5) und (6) in mm/s an den Robotino gesendet. Diese Umrechnung der einzelnen Motorgeschwindigkeiten findet in der Simulation gleichermaßen in dem EIA-232 Treiber Block statt (siehe Abbildung 5.10). Dabei wird in der Embedded Matlab Function „*Omnidirektion*“ eine Matrix mit den zugehörigen Umrechnungsformeln (1), (2) und (3) verwendet. Die Ausgabe der einzelnen Motorgeschwindigkeiten wird anschließend in mm/s mit den jeweiligen Faktoren gewandelt und die Informationen an die Robotinohardware weitergeleitet.

5.6.1.2 Robotino Hardware

Die Robotinohardware setzt die ankommenden Informationen der Motorgeschwindigkeiten auf eine genau umgekehrte Weise um, als der Treiber der seriellen Schnittstelle (siehe Abbildung 5.10). Die einzelnen Motorgeschwindigkeiten werden vektoriell zu einer Roboter Fahrrichtung zusammengesetzt, die über eine Inverse Matrix mit den Umrechnungsformeln (1), (2) und (3), im Block „*Omnidirektion*“, bestimmt werden. Wegen unterschiedlicher Koordinationsdefinitionen der Webcambilder und der Roboterbewegung, wird das Vorzeichen von Vx und Vy gedreht und somit die Bewegungsrichtung an die Bildkoordination angepasst. Dieser Hintergrund wird auch in dem Robotino Blockausgang berücksichtigt, indem die Vx und Vy Ausgänge vertauscht werden. Die Berechnung der aktuellen Position wird durch die Integration der Geschwindigkeiten simuliert. Durch die Initialisierung der Integratoren werden Startpositionen der simulierten Regelung vorgegeben.

5.6.2 Simulation der einfachen Punktregelung

In der Simulation der einfachen Punktregelung wird der Robotino durch mathematische Algorithmen beschrieben und als Regelstrecke in einem Regelkreis eingebaut. Der reale Roboter bekommt als Eingangssignal drei Geschwindigkeiten der einzelnen Motoren, die vektoriell zu einer omnidirektionalen Bewegungsrichtung zusammengesetzt werden. Als Antwort auf die Eingangssignale liefert er über die angeschlossene Webcam die aktuellen Positionen. Dieses integrale Verhalten wird als Regelstrecke, in Kombination mit einem P-Regler, in einem geschlossenen Regelkreis unter Matlab/Simulink realisiert, um damit simuliert den Robotino stationär auf die Führungsgröße zu regeln. Das zugehörige Simulinkmodell wird in der nachfolgenden Abbildung 5.11 dargestellt:

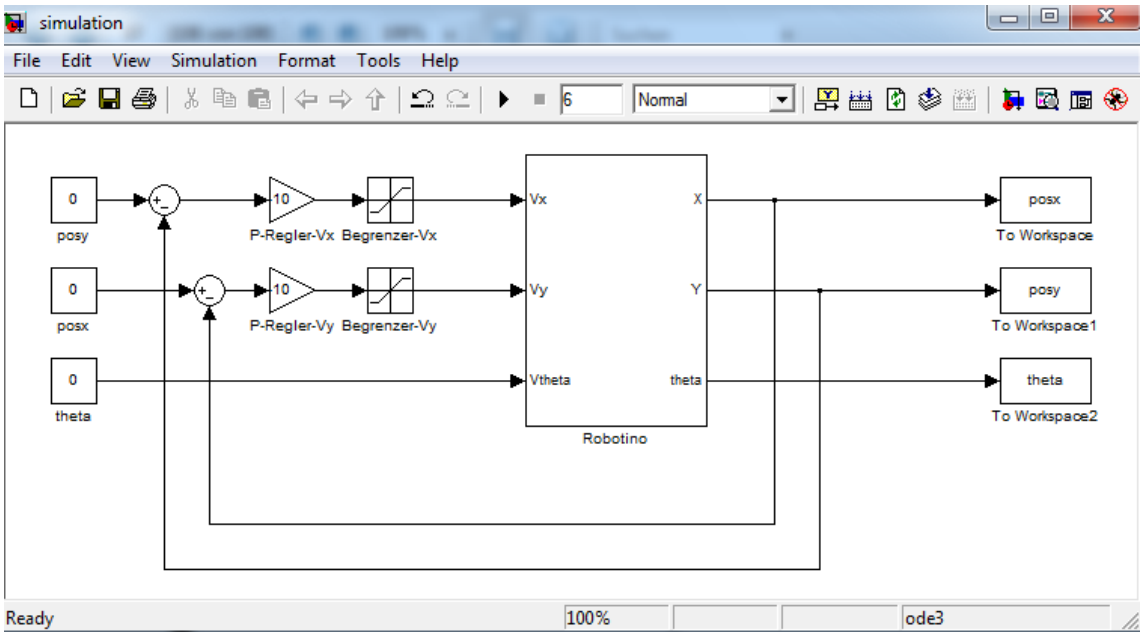

Abbildung 5.11: Simulationsaufbau der einfachen Punktregelung

In dem obigen Simulinkmodell aus der Abbildung 5.11 wird als Führungsgröße die X-Position=0 und die Y-Position=0 gewählt. Die Rotation wird außer acht gelassen, da eine Winkelbestimmung mit dem einfachen Merkmal noch nicht realisierbar ist. Die Regeldifferenz zwischen dem Sollwert und dem Istwert wird jeweils durch einen P-Regler mit dem Faktor „*-10*“ verstärkt. Die negative Verstärkung kommt durch unterschiedliche Richtungs- und Achsendefinitionen der Kamerabilder sowie der zugehörigen Robotinobewegung zustande. Diesbezüglich werden auch die Regeldifferenzen vertauscht. Dieser Zusammenhang wird bei der Realisierung der realen Regelfahrt deutlicher. Die Eingangssignale der Vx und der Vy Geschwindigkeiten der Regelstrecke werden auf -300 mm/s bis 300 mm/s begrenzt, um dadurch eine unzulässige Werteübergabe an den realen Roboter zu verhindern. Bei den kompletten Regelabläufen wird ein Signalintervall von 200 ms vorgesehen, in dem alle Rechenoperationen abgeschlossen sein müssen. Dieses Zeitverhalten wird im Simulinkmodell ebenfalls durch „*Simulation* → *Configuration Parameters*“ unter dem Abschnitt „*Solver options*“ über die Auswahl

- **Type:** Fixed-step
- **Fixed-step size (fundamental sample time):** 0.2

angeglichen. Damit das Simulationsverhalten untersucht werden kann, werden die Ausgangswerte des simulierten Regelkreises an Matlab übergeben. Nach einer Simulationszeit von 10 Sekunden und den Startwerten X=80 und Y=60 werden die simulierten Werte und damit das Regelverhalten der Punktregelung des Robotinos mit dem „*simplot.m M-File*[52] in dem nachfolgenden Plot 5.12 zusammengefasst:

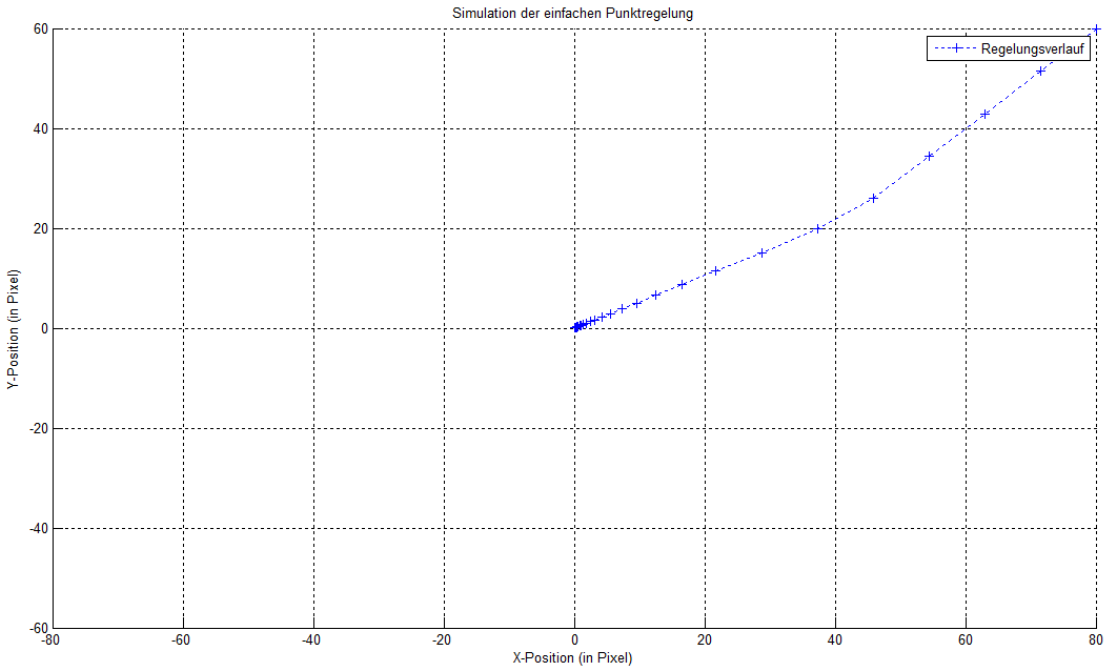

Abbildung 5.12: Simulation der einfachen Punktregelung

[52] Funktionsdatei von Matlab

5.6.2.1 Analyse des Simulationsverhaltens

In der obigen Abbildung 5.12 ist zu erkennen, dass die Position von X=80 und Y=60 auf den stationären Wert X=0 und Y=0 geregelt wird. Mittels der gesetzten Zeitpunkte wird es deutlich, dass die Geschwindigkeit exponentiell bei dem Regelverlauf zur Endposition abnimmt. Dieses Zeitverhalten wird bei einem PI-Regelkreis erwartet. Weiterhin ist in dem Regelverlauf ein Knick, also eine minimale Richtungsänderung sichtbar. Für diese Änderung sind die Begrenzungen der Geschwindigkeiten verantwortlich, die am Anfang der Regelverlaufs an ihre Maxima kommen. Insgesamt ist die Simulation wie erwartet abgelaufen und infolgedessen können die Regelparameter in einer realen Punktregelungsfahrt verwendet werden.

5.6.3 Reale Regelfahrt

Bei der realen Regelfahrt der einfachen Punktregelung wird zuerst das Simulinkmodell aus der Abbildung 5.1 um eine Positionsbestimmung des Robotinos erweitert. Weiterhin wird ein Regelkreis mit den gleichen Regelparametern wie bei der zugehörigen Simulation aufgebaut, in dem die gelieferten Positionsdaten aus den Webcambildern einfließen. Das entsprechende Simulinkmodell sieht wie folgt aus:

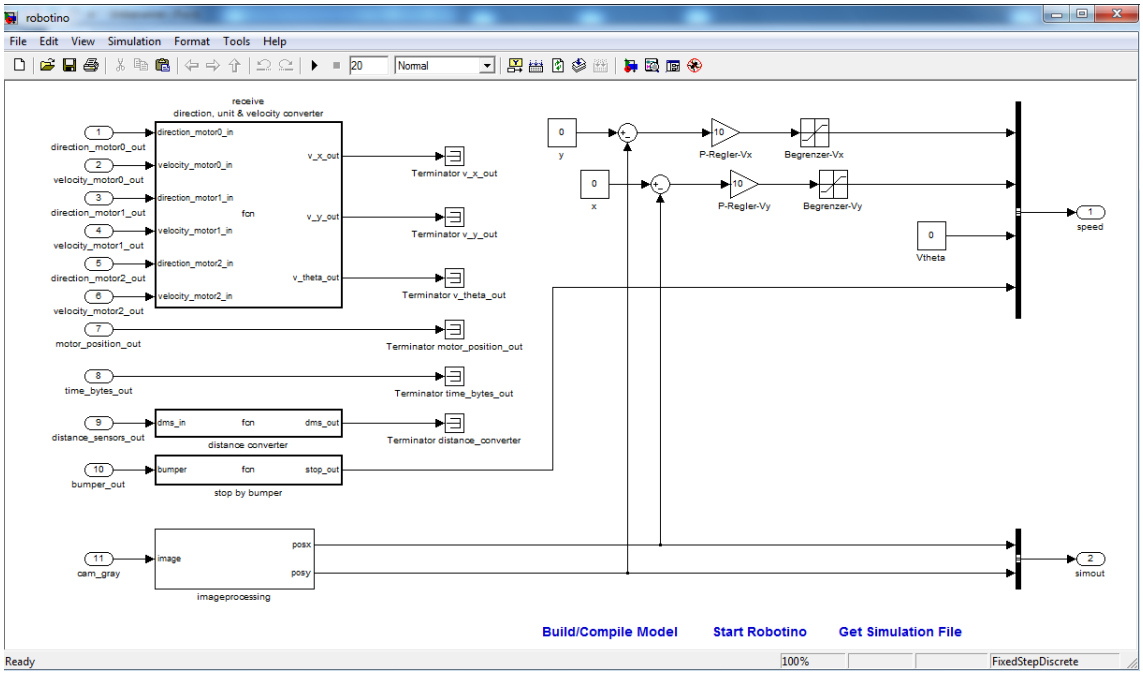

Abbildung 5.13: Simulinkmodell der realen Regelungsfahrt für die einfache Punktregelung

Das obige Simulinkmodell aus der Abbildung 5.13 wird mit dem RTW Embedded Coder in C-Codes umgewandelt, auf den Robotino übertragen, dort kompiliert und ausgeführt. Für die anschließende Analyse dieser Fahrt werden über die ert_main.c Datei die aktuellen

Positionen des Roboters in Abhängigkeit der Zeit abgespeichert und am Ende der Regelfahrt unter Matlab mit der zugehörigen Simulation verglichen.

5.6.3.1 Positionsberechnung

Die Webcam des Robotinos führt Graustufenbilder des einfachen Bezugspunkts in das Simulinkmodell aus der Abbildung 5.13 ein. Diese Bilder werden in dem Block „*Imageprocessing*“ in zwei Funktionen bearbeitet, sodass eine momentane Position des Robotinos bestimmt werden kann. Für die Bestimmung der Position wird in der ersten Funktion das Graustufenbild binarisiert. Dazu wird ein Graustufenwert von 250 gewählt, bei dem alle höheren Grauwerte auf 1 und alle darunterliegenden Grauwerte auf 0 gesetzt werden. Das binarisierte Bild besteht folglich aus nur schwarzen und weißen Pixel, wobei die weißen Pixel nur von den LED's des Bezugspunkts kommen sollten. In der nachfolgenden, zweiten Funktion werden alle weißen Pixelpositionen separat in der X- und Y-Bildrichtung bestimmt und aus denen der Mittelwert gebildet. Mit diesem Vorgehen wird die Mitte des Bezugspunkts auf dem Webcambild ermittelt. Über die Differenz der bestimmten Mitte des Bezugspunkts und der Bildmitte wird letztendlich die Lage des Roboters berechnet und als Istwert in den Regelkreis eingebunden. In der nachfolgenden Abbildung wird dieser Verlauf visualisiert:

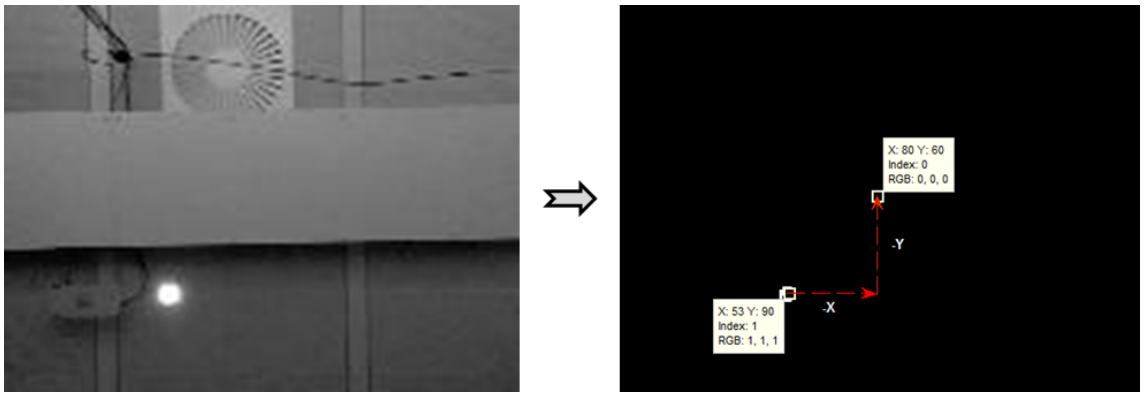

Abbildung 5.14: Bestimmung der Robotinoposition im Binärbild

Bei diesem Verfahren fließen entstandene Pixelfehler durch Lichtreflektionen in die Durchschnittsberechnung der Mitte des Bezugspunkts mit ein und verfälschen somit die Koordination des Robotinos.

5.6.3.2 Rückgabewerte der Regelfahrt

Zur Analyse des realen Regelverlaufs werden die Istwerte des Regelkreises aus dem Simulinkmodell 5.13 über den Ausgangsport „*simout*“ kontinuierlich hinausgeführt und während des Programmablaufs abgespeichert. Dafür wird in ert_main.c zu Beginn des Programms die Datei „*simout.csv*“ erzeugt, in die die Programmlaufzeiten, die aktuelle X- und die Y Position des Robotinos hineingeschrieben werden. Damit diese Rückgabewerte von Matlab

gelesen werden können, wird eine Datei mit der Endung „*.csv*“ benutzt. Diese Formate werden unter Matlab für Importierungen externer Tabellenwerte verwendet. Der Inhalt dieser Dateien muss eine vordefinierte Syntax haben, bei der die Spalten üblicherweise durch ein Leerzeichen definiert und die Zeilen anhand eines Zeilenumbruchs erkannt werden. Nach Ablauf der Regelfahrt kann die erstellte Datei mit den Rückgabewerten letztendlich von dem Robotino heruntergeladen, in Matlab durch einen Doppelklick importiert und anhand von Plots untersucht werden. Diese Abspeicherung der Rückgabewerte wird in allen nachfolgenden Regelfahrten simultan durchgeführt.

5.6.3.3 Unterschiedliche Koordinationsdefinitionen

Die Pixel in einem digitalen Bild werden von der oberen linken Ecke aufgebaut, sodass nach unten eine zunehmende Y-Achse und nach rechts eine zunehmende X-Achse definiert wird. Die Robotinogeschwindigkeiten wurden in den vorherigen Kapiteln so definiert, dass die X-Achse die Vorwärtsgeschwindigkeit und die Y-Achse die Seitwärtsgeschwindigkeit beschreibt. Der Robotino fährt bei einem positiven Vx-Wert nach vorne und bei einem positive Vy-Wert nach links (siehe Abbildung 5.15).

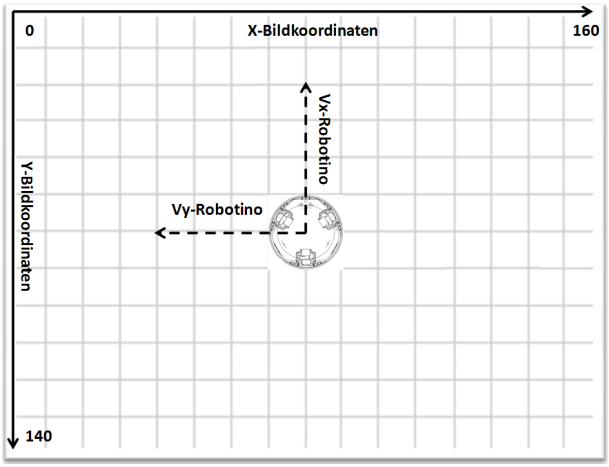

Abbildung 5.15: Koordinationsdefinitionen der Webcambilder und des Robotinos

Um diese unterschiedlichen Koordinationen anzupassen, werden die Vorzeichen der Robotinogeschwindigkeiten und deren Richtungen in den Simulationsmodellen sowie in den Simulinkmodellen für die reale Regelfahrten getauscht.

5.6.3.4 Analyse des realen Regelverhaltens

Zur Analyse des Regelverhaltens wird eine reale Fahrt durchgeführt, die jeweiligen Werte in Matlab importiert und anhand eines Plots mit dem zugehörigen Simulationsmodell verglichen. Die nachfolgende Abbildung zeigt diese Gegenüberstellung:

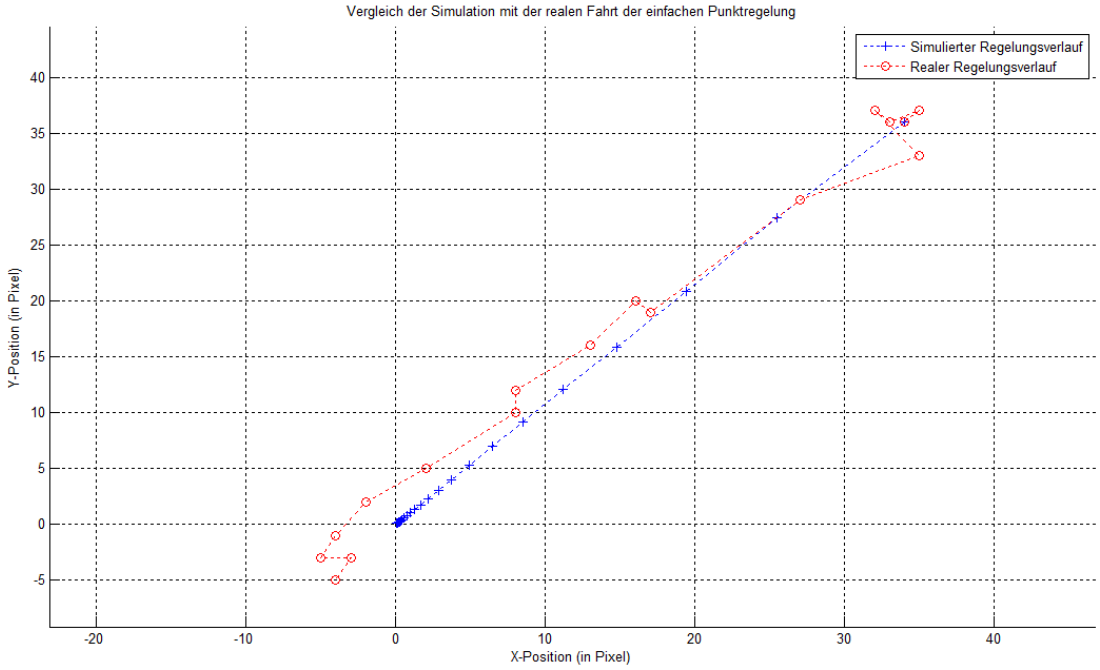

Abbildung 5.16: Vergleich der Simulation mit der realen Fahrt der einfachen Punktregelung

In der Abbildung 5.16 ist zu erkennen, dass die reale Regelfahrt sich dem stationären Endpunkt nähert. Beim Programmstart wird einige Zeit benötigt, um die Treiber und die zugehörige Hardware zu initialisieren. Diese aufgewendete Zeit ist ebenfalls in dem Startpunkt X=34 und Y=36 bei der realen Regelfahrt sichtbar. Ein weiterer Aspekt der Ungenauigkeit beim Start ist das Anfahrverhalten des Roboters. Dabei entsteht ein Ruck des ganzen Systems, der die Ausrichtung des Robotinos sowie die Position verändert. Bei der geringen Auflösung der Webcam wird jede kleinste Positionsänderung deutlich im realen Regelverlauf sichtbar. Dies ist auch beim Erreichen des stationären Endpunkts erkennbar. Der Regelverlauf weist einen Überschwinger auf und bleibt dann einige Pixel neben dem stationären Endwert stehen. Weiterhin können bei dem realen Regelverlauf fehlinterpretierte Lichtreflektionen die Robotinoposition verfälschen. Insgesamt wurde jedoch ein ausreichendes Ergebnis erzielt, das mit der digitalen Bildverarbeitung eventuell noch verbessert werden kann.

5.7 Punktregelung mit digitaler Bildverarbeitung

Die Punktregelung mit digitaler Bildverarbeitung soll den gleichen Regelkreis und damit die gleichen Regelfunktion haben, wie die einfache Punktregelung. In diesem Abschnitt wird jedoch die Positionsbestimmung des Robotinos durch eine komplexere Bildverarbeitung verbessert. Für den Vergleich der späteren realen Regelfahrt kann das gleiche Simulationsmodell aus der Abbildung 5.11 verwendet werden, wie bei der einfachen Punktregelung.

5.7.1 Digitale Bildverarbeitung

Für die Punktregelung mit digitaler Bildverarbeitung wird das Simulinkmodell 5.13 als Basis genommen und der Block „*imageprocessing*“ mit mehreren Bildverarbeitungsfunktionen ergänzt. Bei dieser Bezugspunkterkennung werden mehrere, zusammenhängende Pixelobjekte aus dem aufgenommen Bild nach ihren Pixelvolumina untersucht und dadurch entschieden, ob es sich dabei um den Bezugspunkt handelt oder um einen Fehler durch Lichtreflektionen. Die zugehörigen Funktionen aus dem „*imageprocessing*“ Block werden anhand der nachfolgenden Abbildung erläutert:

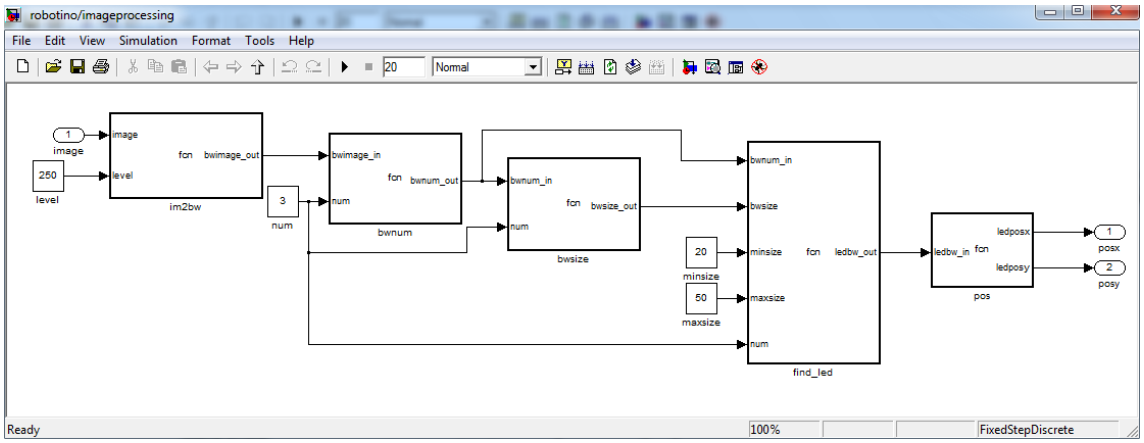

Abbildung 5.17: Funktionen der digitalen Bildverarbeitung zur Erkennung des Bezugspunkts

im2bw: In dieser Funktion wird das eingegangene Graustufenbild binarisiert. Es werden alle Graustufenwerte, die höher sind als 249 auf 1 gesetzt und alle darunterliegenden Grauwerte auf 0. Das Ergebnis der Binarisierung wird an die nächste Funktion weitergeleitet.

bwnum: Durch die Binarisierung des Graustufenbildes enthält jetzt das Ergebnis im schlimmsten Fall mehrere weiße Pixel, die im Bild verstreut sind. Weiße Pixel, die sich nebeneinander befinden, werden als ein zusammengehöriges Objekt definiert. In der bwnum Funktion wird die Nachbarschaft jedes weißen Pixels untersucht und entschieden, ob es zu einem größeren Objekt gehört oder nicht. Als Pixelachbern gelten hier Pixel, die sich oben, unten, recht und links von dem jeweiligen weißen Pixel befinden. Weiterhin bekommen in dieser Funktion alle benachbarten Pixel dieselbe Nummer, sodass alle Objekte in dem binarisierten Bild nummeriert werden. In der nachfolgenden Abbildung wird ein Beispiel mit drei Objekten gezeigt, in einer invertierten Farbdarstellung:

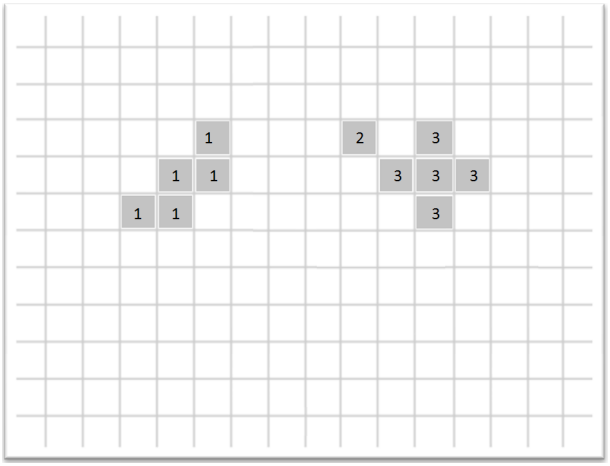

Abbildung 5.18: Nummerierung der Objekte im binarisierten Bild

Im Bezug auf den großen Rechenaufwand dieser Funktion, wird durch die Eingangskonstante „*num*“ eine Nummerierung von maximal drei Objekten gesetzt. Die restlichen Objekte werden aus dem binarisierten Bild entfernt. Diese Begrenzung fließt auch in die nachfolgenden Funktionen ein. In den meisten Fällen werden bei der Aufnahme des Bezugspunkts nicht mehr als zwei Objekte nach der Binarisierung auftauchen. Das Bild mit den nummerierten Objekten wird an die nächst Funktion weitergeleitet.

bwsize: Die nummerierten Objekte werden in der bwsize Funktion auf die Pixelanzahl untersucht. In diesem Block wird gezählt, wie viel Pixel jedes einzelne Objekt beinhaltet. Diese Volumenbestimmung ist ebenfalls auf drei Objekte begrenzt. Die Ergebnisse werden an die nächsten Funktionen weitergeleitet.

find_led: In diesem Block wird nach dem Bezugspunkt gesucht. Hierbei fließt das nummerierte Binärbild und die Volumengrößen der jeweiligen Objekte des Bildes ein. Anhand von zwei Konstanten wird eine minimale und maximale Größe des Bezugspunkts definiert. Das Pixelvolumen des Bezugspunkts im Bild wurde durch eine Testaufnahme aus der nachfolgenden Abbildung ermittelt:

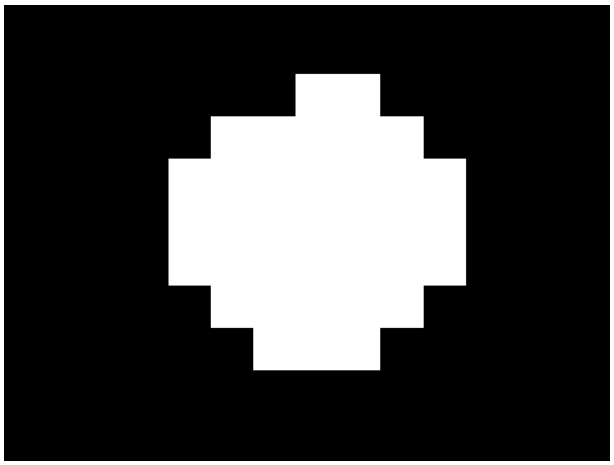

Abbildung 5.19: Pixelvolumen des einfachen Bezugspunkts

Das Pixelvolumen des Bezugspunkts aus dem vergrößerten Binärbild 5.19 enthält 36 Pixel. Daher wird das minimale Volumen beim Vergleich auf 20 Pixel und das maximale auf 50 Pixel gesetzt. Der Bezugspunkt in dem binarisierten Bild wird folglich durch eine Übereinstimmung des definierten Pixelvolumens mit dem vorhandenen Objektvolumina aus der Aufnahme bestimmt. Die restlichen Objekte werden entfernt und das binarisierte Bild mit dem übriggebliebenen Bezugspunkt an die letzte Funktion weitergeleitet.

pos: In der letzten Funktion wird letztendlich die Mitte des Bezugspunkts, durch Bildung der Mittelwerte der Positionen der weißen Pixel in der X- und Y-Richtung im Bild bestimmt. Durch die Differenz der bestimmten Mitte des Bezugspunkts mit dem Bildmittelpunkt wird die Position des Roboters berechnet, die dann in den Regelkreis als Istwert einfließt.

Bei dieser digitalen Bildverarbeitung werden, im Vergleich zur einfachen Punktregelung, keine Fehler durch Lichtreflektionen in die Berechnung der Mitte des Bezugspunkts einfließen. Infolgedessen wird die Positionsbestimmung des Robotinos für die anstehende Regelfahrt verbessert.

5.7.2 Analyse des realen Reglverhaltens

Zur Analyse des Regelverhaltens wird eine reale Fahrt durchgeführt, die jeweiligen Werte in Matlab importiert und anhand eines Plots mit dem zugehörigen Simulationsmodell verglichen. Die nachfolgende Abbildung zeigt diese Gegenüberstellung:

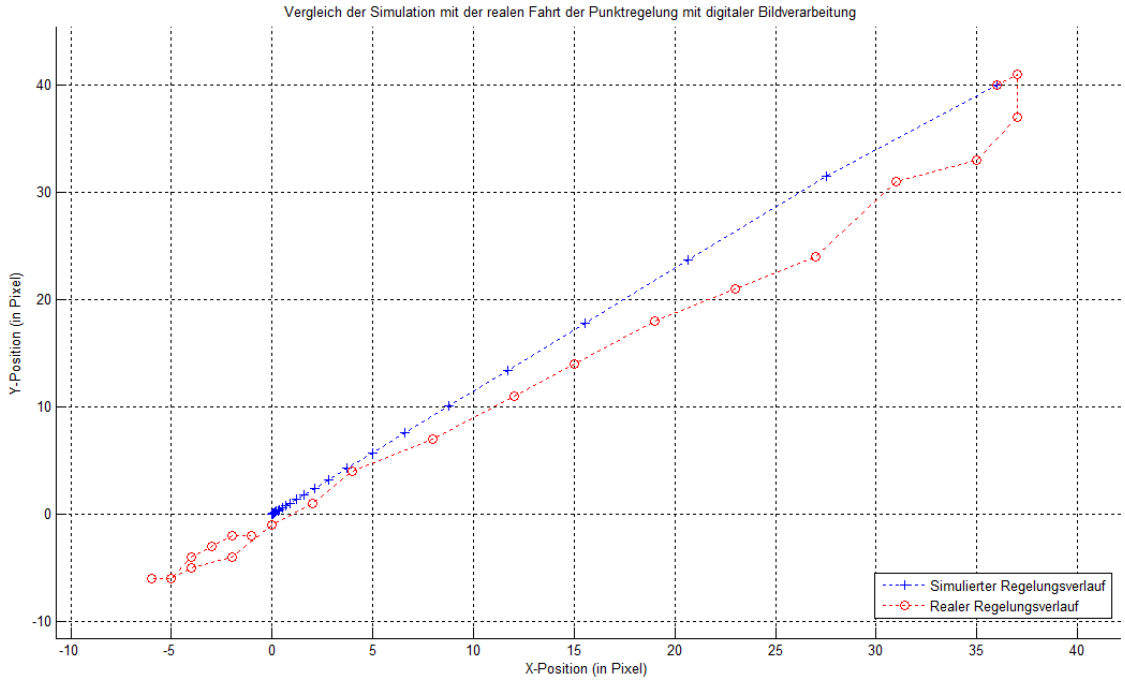

Abbildung 5.20: Vergleich der Simulation mit der realen Fahrt der Punktregelung mit digitaler Bildverarbeitung

In den Plots in der Abbildung 5.20 ist zu erkennen, dass die reale Regelfahrt sich dem stationären Endpunkt X=0 und Y=0 nähert. In dem Plot ist ebenfalls ein Anfangsruck des Systems beim Start der Regelfahrt wahrnehmbar. Der Verlauf der Fahrt ist insgesamt etwas linearer, als bei der Regelfahrt ohne digitaler Bildverarbeitung (siehe Abbildung 5.16). Am Ende des Prozesses entsteht ein Überschwinger, der sofort ausgeregelt wird. Das PI-Verhalten der Regelstrecke, bei dem die Fahrt zum stationären Endpunkt hin verlangsamt wird, ist auch bei diesem Regelverlauf gegeben. Insgesamt wurde ein gutes Ergebnis erzielt, bei dem die Identifizierung des Bezugspunkts mit der digitalen Bildverarbeitung sicherer ist. Diese Innovation kann für den weiterentwickelten Bezugspunkt verwendet werden, bei dem drei LED's in Form eines Dreiecks erkannt werden müssen, um dadurch die Position und die Rotationslage des Robotinos bestimmen zu können.

5.8 Rotierende Vierpunktregelung

Die erfolgreichen Integrationsregelfahrten mit dem einfachen Bezugspunkt ermöglichen eine Erweiterung des Verfahrens. Auf Basis des bereits verwendeten Simulinkmodells aus der Punktregelung mit digitaler Bildverarbeitung wird der weiterentwickelte Bezugspunkt eingesetzt, um eine rotationsabhängige Regelfahrt mit der USB Webcam zu realisieren. In diesem Zusammenhang wird durch das neue LED Muster auf der Decke die Rotationslage des Robotinos bestimmt und daraus neue Vektorgeschwindigkeiten berechnet, sodass der Robotino sich auf dem direkten Wege der Führungsgröße bei der realen Regelfahrt stationär nähert. Für einen etwas komplexeren Prozess werden die Führungsgrößen umgeschaltet,

sobald der Robotino sich der jeweiligen Sollgröße in einem bestimmten Bereich angenähert hat. Eine Umschaltung zwischen vier Sollwerten wird dafür sorgen, dass der Roboter sich in einem Viereck bewegt. Weiterhin wird dem Robotino eine feste Rotationsgeschwindigkeit gesetzt, sodass eine rotierende Vierpunktregelung unter dem weiterentwickelten Bezugspunkt realisiert wird. Zu Analysezwecken wird zuerst dieses Vorhaben mit Matlab/Simulink simuliert und im Anschluss darauf mit der realen Fahrt verglichen.

5.8.1 Erweiterung der mathematischen Beschreibung des Robotinos

Die mathematische Beschreibung des Robotinos aus der Abbildung 5.10 wird um die Erfassung der Rotationslage des Roboters in Hinsicht auf den erweiterten Bezugspunkt ergänzt. Der Robotino bekommt als Eingang die Geschwindigkeiten vorgegeben und antwortet darauf zeitabhängig mit der aktuellen Position sowie der Rotationslage, die über die Webcam berechnet werden. Zu diesem Zweck wird zuerst der Zusammenhang zwischen den rotierenden Ebenen der Vektorgeschwindigkeiten und den Bezugskoordinaten aus der nachfolgenden Abbildung erläutert:

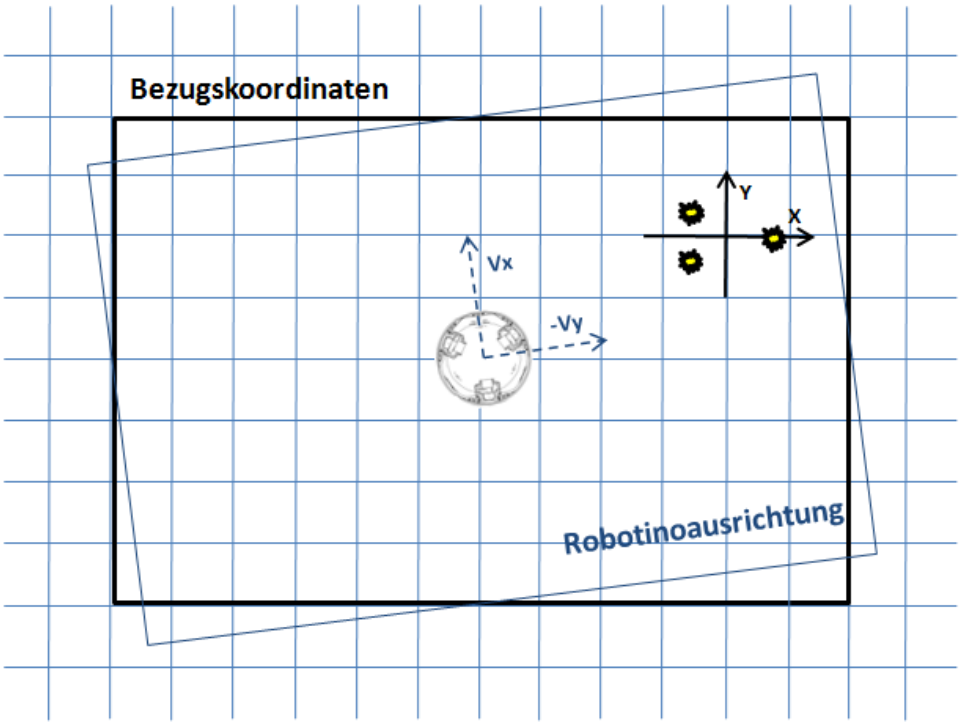

Abbildung 5.21: Zusammenhang der mathematischen Robotinorotation

In oberen Abbildung 5.21 wird der mathematische Hintergrund der Bilderfassung bei einer konstanten Rotation des Roboters grafisch dargestellt. Von außen gesehen werden bei diesem Verlauf in der Realität die X-, Y-Abstandskoordinaten zum Ursprung nicht verändert. Bei den zeitkontinuierlichen Bildaufnahmen der Webcam sieht dieses Verhalten anders aus. Der Bezugspunkt wandert bei einer positiven Drehrichtung des Roboters, in einem festen

Radius, in entgegengesetzter Richtung um die Bildmitte herum. Dieses Verhalten wird in der Simulation durch die Vektorrotation nachgebildet. Weiterhin muss die Ausrichtung der Vektorgeschwindigkeiten Vx und Vy des Roboters bei diesem Prozess mitberücksichtigt werden. Die Vektorgeschwindigkeiten drehen sich zusammen mit dem Roboter, sodass diese im Bezug auf die Bezugskoordinaten um die positive Drehrichtung in der Simulation rotiert werden. Am einfachsten kann sich dieser Zusammenhang so vorgestellt werden, dass der Robotino sich auf eine Stelle mit den entsprechenden Ausrichtungen der Vektorgeschwindigkeiten dreht. Der Bezugspunkt wandert dabei in einem festen Abstand um die Bildmitte herum. Das zugehörige Simulinkmodell ist in der nachfolgenden Abbildung dargestellt:

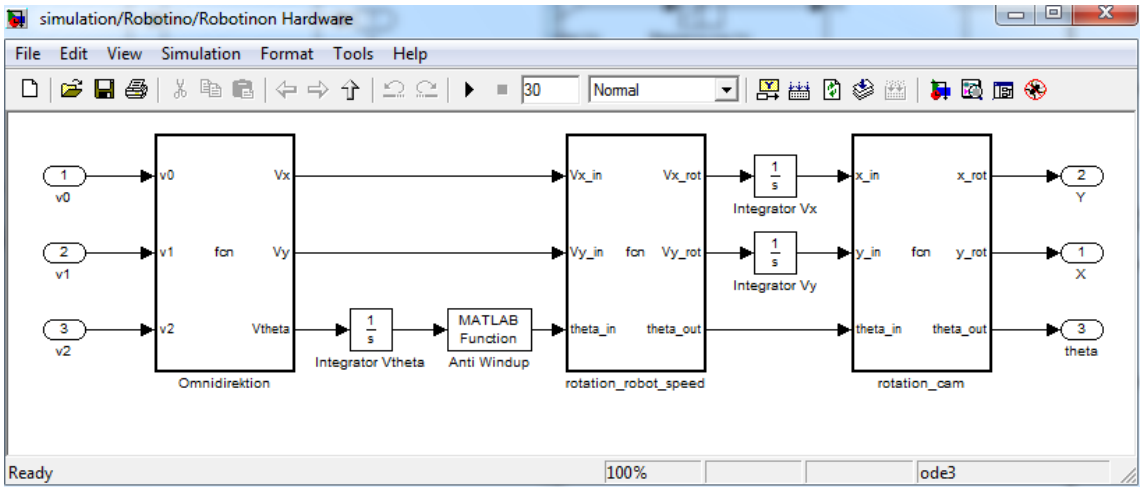

Abbildung 5.22: Winkelabhängige mathematische Beschreibung des Robotinos

Wegen einer simulierten Integration der Winkelgeschwindigkeit steigt bei einem konstanten Vtheta Wert die Winkelausgabe ins Unendliche. Die reale Rotationslage des Roboters ist von -π bis π begrenzt. Folglich wird in der Simulation aus der Abbildung 5.22 in der Anti Windup Funktion die Winkelausgabe mathematisch begrenzt und damit das Verhalten der Realität angepasst.

5.8.1.1 Vektorrotation

Der Mathematische Hintergrund für die Vektorrotation für die Berechnung der Robotinoausrichtung und seiner Koordinationslage wird aus der nachfolgenden Abbildung hergeleitet:

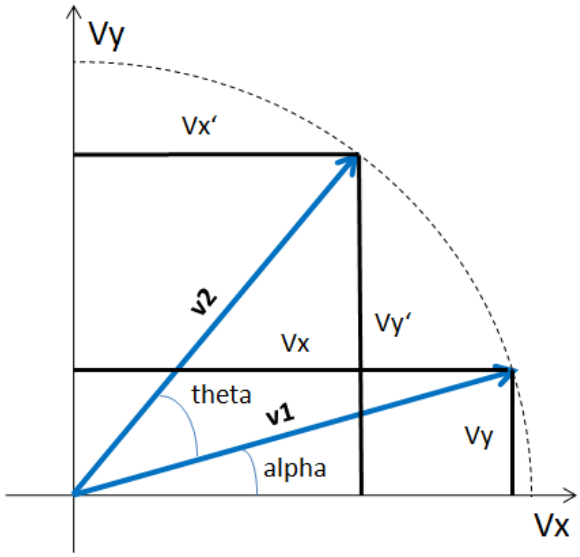

Abbildung 5.23: Herleitung der Vektorrotation

Bei der Vektorrotation aus der Abbildung 5.23 ist zu erkennen, dass die resultierenden Vektoren v1 und v2 den gleichen Betrag haben. Daraus folgt:

$$v1 = v2 \tag{11}$$

Der Vektor v1 wird mit nachfolgenden geometrischen Komponenten beschrieben:

$$Vx = v1 \cdot cos(\alpha) \quad = \quad v1 = \frac{Vx}{cos(\alpha)} \tag{12}$$

$$Vy = v1 \cdot sin(\alpha) \quad = \quad v1 = \frac{Vy}{sin(\alpha)} \tag{13}$$

Aus der Gleichung (11) folgt für die geometrischen Komponenten von v2:

$$Vx' = v1 \cdot cos(\alpha + \theta) = v1 \cdot (cos(\alpha) \cdot cos(\theta) - sin(\alpha) \cdot sin(\theta)) \tag{14}$$

$$Vy' = v1 \cdot sin(\alpha + \theta) = v1 \cdot (sin(\alpha) \cdot cos(\theta) + cos(\alpha) \cdot sin(\theta)) \tag{15}$$

Die Geleichungen (12) und (13) eingesetzt in (14) und (15) ergibt letztendlich den Zusammenhang für die Vektorrotation von v1 um den Winkel theta:

$$Vx' = Vx \cdot cos(\theta) - Vy \cdot sin(\theta) \tag{16}$$

$$Vy' = Vy \cdot cos(\theta) + Vx \cdot sin(\theta) \tag{17}$$

Die hergeleiteten Formeln werden in dem Robotino Simulinkmodell aus der Abbildung 5.22 in den Blöcken „*rotation_robot_speed*“ und „*rotation_cam eingesetzt*“ verwendet, um eine simulierte Drehung des Robotinos nachzubilden.

5.8.2 Simulation der rotierenden Vierpunktregelung

Die Simulation der rotierenden Vierpunktregelung wird auf Basis der simulierten Punktregelung realisiert. Bei dieser Simulation wird das Simulinkmodell aus der Abbildung 5.11 um die Winkelabhängige mathematische Beschreibung des Robotinos erweitert. Die Schrittzeit der Simulation bleibt bestehen, sodass die Echtzeit von 200 ms der realen Fahrt auch hier berücksichtigt wird. Weiterhin wird eine Funktion entwickelt, die für die Umschaltung der Führungsgrößen sorgt. Durch die Rotation des Robotinos werden zu jedem Simulationsschritt neue Vx und Vy Geschwindigkeiten berechnet, die den Roboter abhängig von dem Rotationswinkel auf dem direkten Weg zur Führungsgröße regeln. Diese Berechnung wird in den Regelkreis der Simulation ebenfalls integriert. Das komplette Simulationsmodell für die rotierende Vierpunktregelung wird wie folgt realisiert:

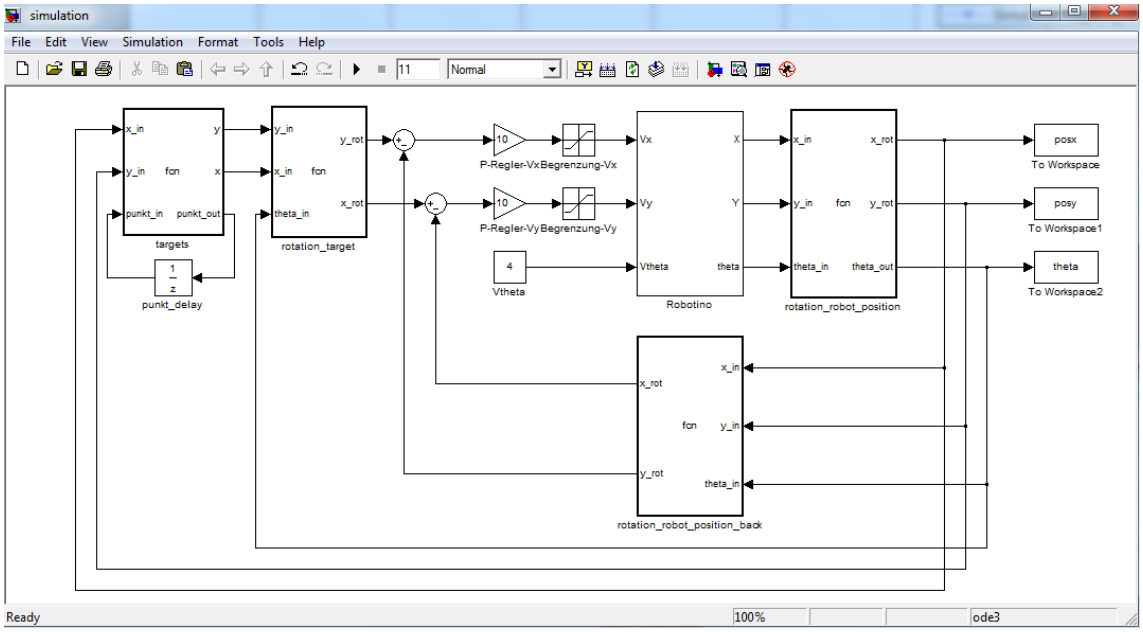

Abbildung 5.24: Simulationsmodell der rotierenden Vierpunktregelung

targets: In dem targets Block werden die vier Führungspunkte definiert sowie deren Umschaltung durchgeführt. Nach dem Erreichen eines Führungswertes in einem Umkreis von zwei Pixel wird der Führungswert mit der „*punk_delay*“ Funktion zum nächsten Ziel umgeschaltet. Diese Funktion agiert wie eine Art Merker, der den Ausgangswert des letzten Simulationsschritts speichert und wieder in den targets Block hineinführt.

rotation_robot_position: In dieser Funktion wird die reale Position des Robotinos über das Kamerabildes ermittelt. Dazu wird in Abhängigkeit der Bezugspunktausrichtung die Lage des Roboters in die negative Richtung mit der Vektorrotation gedreht. Dieser Zusammenhang wird nochmal in der nachfolgenden Abbildung verdeutlicht:

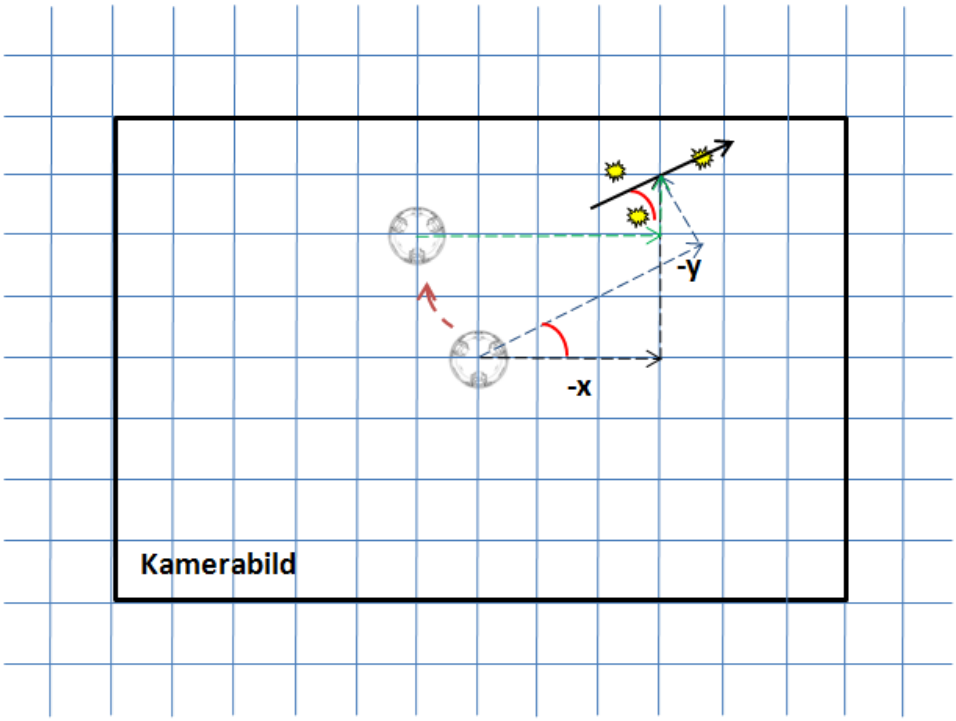

Abbildung 5.25: Ermittlung der realen Position des Robotinos

Die Position des Robotinos wird im Bild über die Pixelentfernung zur Mitte des Bezugspunkts in die X- und Y-Richtung bestimmt (schwarze Kennlinien). Die reale Entfernung (blaue Kennlinien) ist jedoch von der Rotationslage des Bezugspunkts abhängig und stimmt dadurch mit der gemessenen Pixelentfernung nicht überein. Aus diesem Grund wird die Pixelentfernungen in der X- und Y-Richtung um den Ausrichtungswinkel des Bezugspunkts in die negative Richtung gedreht (grüne Kennlinien), sodass die Pixelentfernung im Bild gleich der realen Entfernung entspricht. Diese Bezugsrotation wird auf die gleiche Weise mit den Führungspunkten durchgeführt, jedoch mit einer positiven Drehrichtung.

rotation_target: Die Mitte des Bezugspunkts dient bei der realen sowie der simulierten Regelfahrt als Ursprung der Robotinokoordination. Infolgedessen werden hier die eingehenden Führungsgrößen in Abhängigkeit der Bezugspunktausrichtung mit der Vektorrotation in die positive Richtung gedreht, sodass eine reale Zielposition über das Kamerabild definiert wird.

rotation_robot_position_back: Für die Regelung des Robotinos wird in dem Regelkreis als Istwert die Bildabhängige und nicht die reale Lage des Roboters benötigt. Diesbezüglich wird in dieser Funktion die vorher berechnete reale Position wieder zurückgedreht. Durch die Wanderung des Bezugspunkts um die Bildmitte, bei einer festen Rotationsgeschwindigkeit des Roboters, werden im Bezug auf die Ausrichtung des Robotinos die zugehörigen Regeldifferenzen bestimmt (siehe Abbildung 5.26).

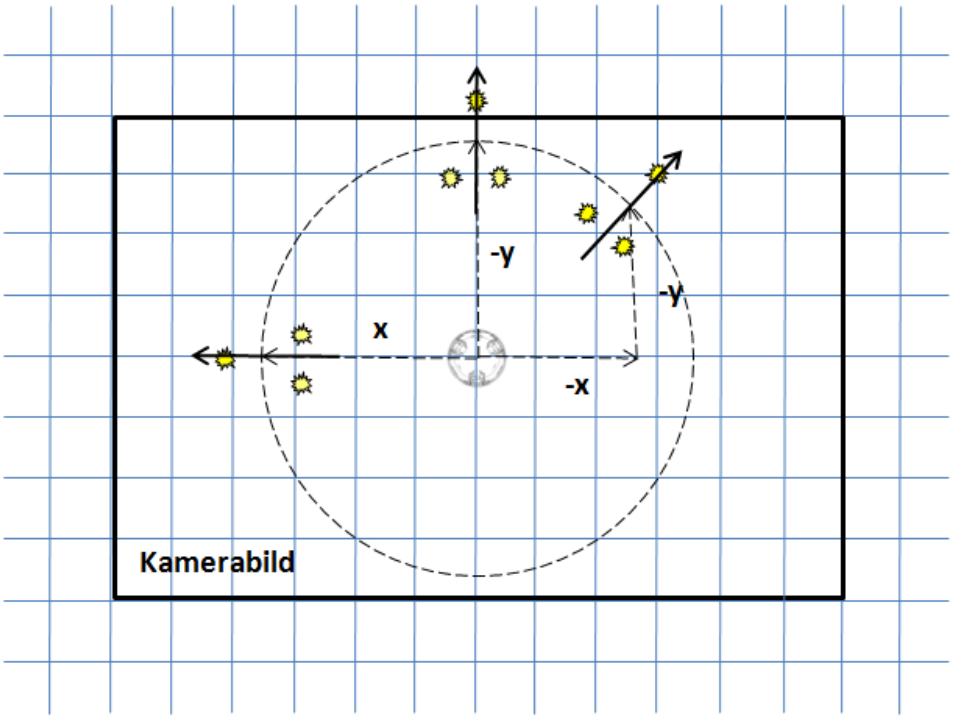

Abbildung 5.26: Berechnung der Regeldifferenz

Bei der Simulation bleiben alle Reglerparameter sowie die Begrenzungen der Geschwindigkeiten im Vergleich zu den vorherigen Simulationen unverändert. Des Weiteren wird eine feste Größe von 4 rad/s für die Rotationsgeschwindigkeit gesetzt, sodass der Robotino sich bei dieser simulierten Regelfahrt mit konstanter Geschwindigkeit im Uhrzeigersinn dreht. Die Wahl der Führungspunkte wurde so gesetzt, dass der simulierte Regelverlauf einen Quadrat bildet. Nach einer Simulationszeit von 11 Sekunden und den Startwerten X=0, Y=0 und theta=0 werden die simulierten Werte und damit das Regelverhalten der rotierenden Vierpunktregelung des Robotinos mit dem simplot.m M-File in dem nachfolgenden Plot 5.27 zusammengefasst:

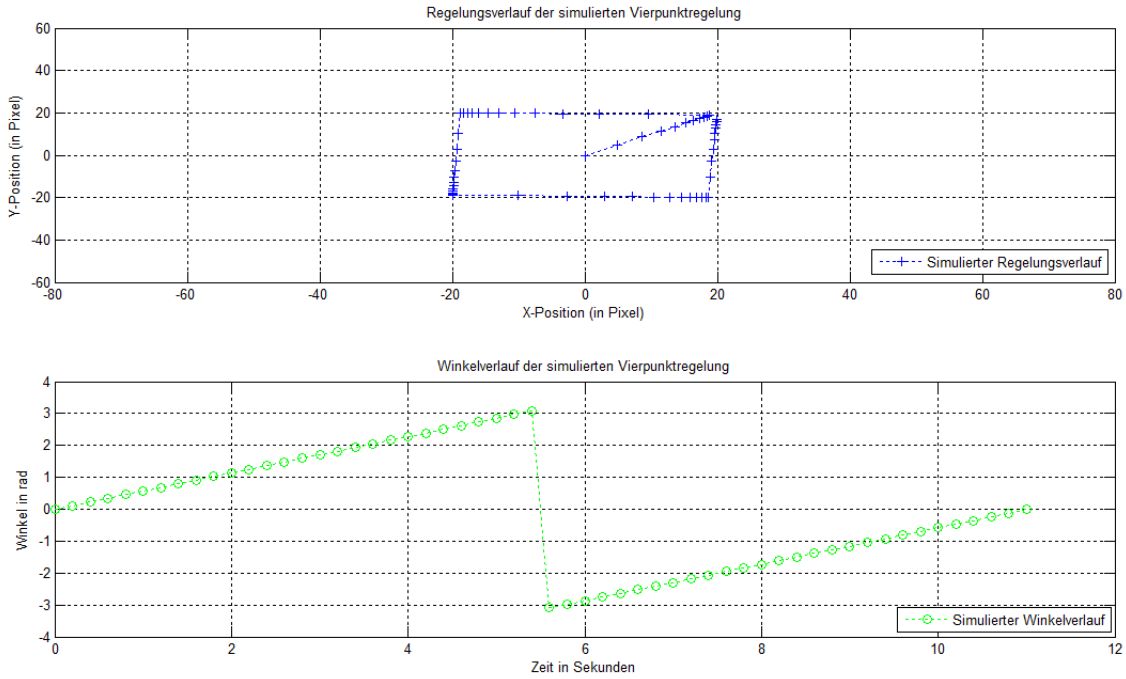

Abbildung 5.27: Simulation der rotierenden Vierpunktregelung

5.8.2.1 Analyse des Simulationsverhaltens

In dem oberen Simulationsplot aus der Abbildung 5.27 ist der simulierte Regelverlauf der rotierenden Vierpunktregelung des Robotinos dargestellt. Die Achsen wurden so gelegt, dass sie die Auflösung des Webcam nachbilden. Im Verlauf dieser Simulation sind die vier Führungspunkte zu sehen und die direkte Regelung des Robotinos auf sie zu. Der Start der Regelfahrt beginnt in dem Ursprung des Koordinatensystems. Weiterhin ist zu erkennen, dass zu jedem Führungspunkt die Geschwindigkeit des Roboters reduziert wird. Dies deutet auf ein PI-Verhalten des Regelkreises, der in der Simulation aus der Abbildung 5.24 realisiert worden ist. Die Ursache für den etwas gekippten Quadratverlauf ist die gesetzte Toleranz beim Umschalten der Führungspunkte. In dem unteren Plot aus der Abbildung 5.27 ist die Ausrichtung des Roboters dieser Regelfahrt zu sehen. Hier ist zu erkennen, dass der Winkel von $-\pi$ bis π begrenzt ist und somit der Realität angepasst. Der Winkel steigt in dem Plot mit einer konstanten Steigung bis π und fällt dann auf $-\pi$ runter, von wo die Winkelsteigung fortgesetzt wird. Dieser Ablauf bildet eine konstante Rotation des Robotinos während der Regelfahrt nach. Insgesamt ist die Simulation wie erwartet abgelaufen und infolgedessen können die Regelparameter in einer realen rotierenden Vierpunktregelung verwendet werden.

5.8.3 Reale Regelfahrt

Bei der realen Regelfahrt der rotierenden Vierpunktregelung wird das Simulinkmodell zur Regelung des realen Robotinos erweitert. An dieser Stelle werden die gleichen Regelparameter sowie die bereits beschriebenen Funktionen zur Erfassung der Rotationslage des Robotinos verwendet, wie aus der Simulation 5.24 des weiterentwickelten Bezugspunkts. Das zugehörige Simulinkmodell ist in der nachfolgenden Abbildung dargestellt:

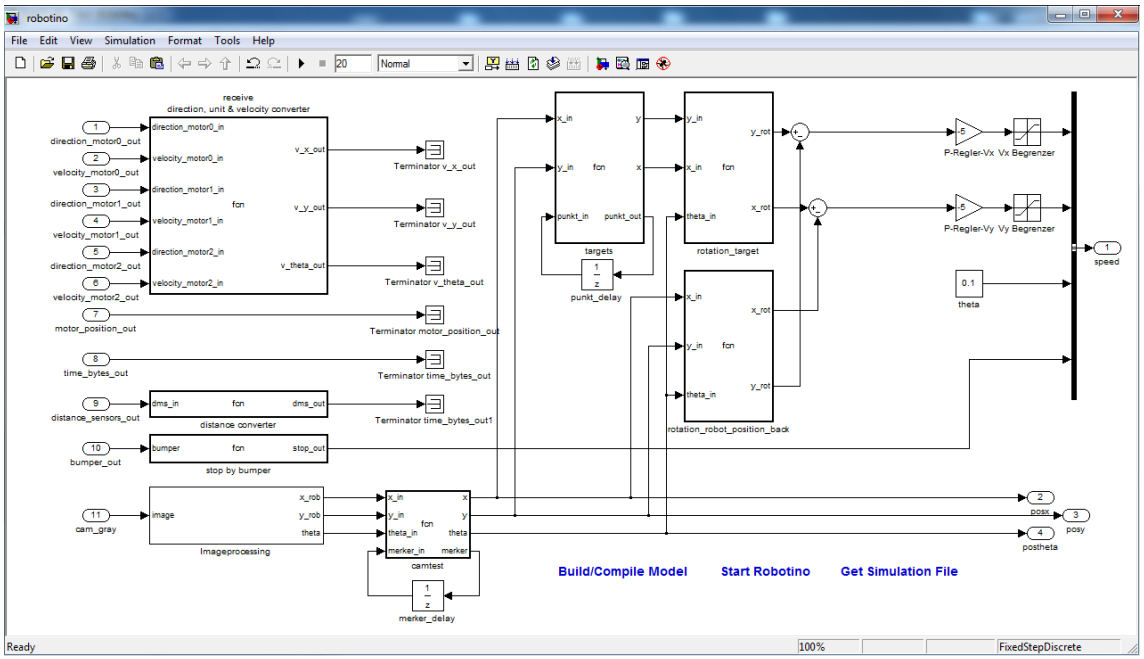

Abbildung 5.28: Simulinkmodell der rotierenden Vierpunktregelung

Zur Koordinationslage des Robotinos wird in diesem Modell ein zusätzlicher Ausgangsport eingefügt, der den Ausrichtungswinkel des Roboters hinausführt. Dieser Winkel wird in dem Funktionsblock „*imageprocessing*“ mit der digitalen Bildverarbeitung über die Kameraaufnahmen des erweiterten Bezugspunkts berechnet. Bei dieser Berechnung kann es vorkommen, dass der Bezugspunkt nicht erkannt wird und infolgedessen die Ausganswerte null herausgegeben werden. In diesem Fall speichert der nachfolgende Funktionsblock „*camtest*“ immer die letzten gültigen Positionswerte und gibt diese, bei einem nicht Erkennen des Bezugspunkts, als die aktuell ermittelten Koordinaten des Robotinos heraus. Auf diese Weise wird der letzte geltende Wert solange benutzt, bis ein neuer berechnet werden kann. Bei dieser Fahrt wird eine Feste Rotationsgeschwindigkeit von 0.1 rad/s gesetzt und die Verstärkung der P-Regler halbiert.

5.8.3.1 Digitale Bildverarbeitung zum weiterentwickelten Bezugspunkt

Die digitale Bildverarbeitung zum weiterentwickelten Bezugspunkt zur Bestimmung der Position und der Ausrichtung des Robotinos wird aus dem Simulinkmodell 5.28 in dem

Block „*imageprocessing*“ durchgeführt. Die beinhaltenden Funktionen werden auf Basis der Erkennung des einfachen Bezugspunkts aus 5.17 erweitert und durch die nachfolgende Abbildung beschrieben:

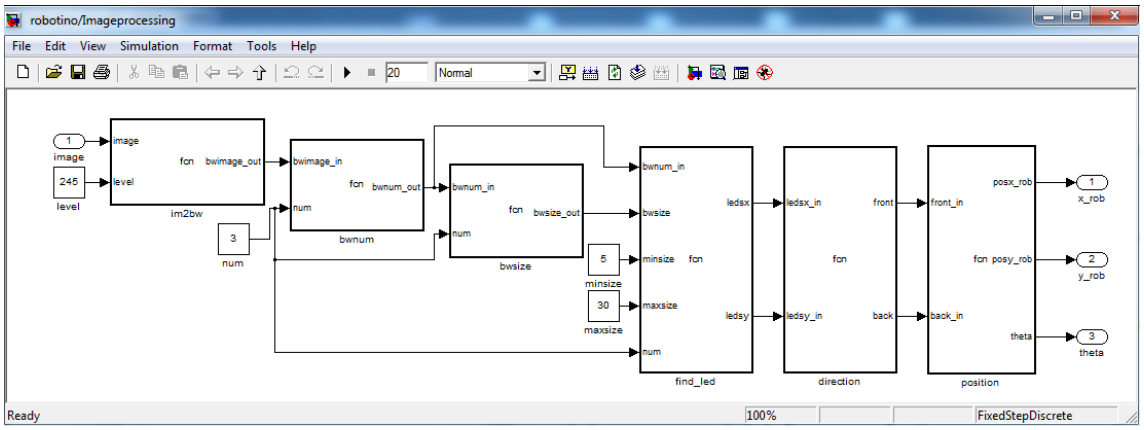

Abbildung 5.29: Digitale Bildverarbeitung der rotierenden Vierpunktregelung

Die ersten drei Funktionen aus dem obigen imageprocessing Block haben dieselben Funktionalitäten, wie aus dem Vorgängermodell 5.17. In diesen wird das aufgenommene Bild des weiterentwickelten Bezugspunkts binarisiert, die zusammengehörenden Pixelobjekte nummeriert und deren Pixelvolumina berechnet. Ein Beispiel dieses Vorgehens kann wie folgt aussehen:

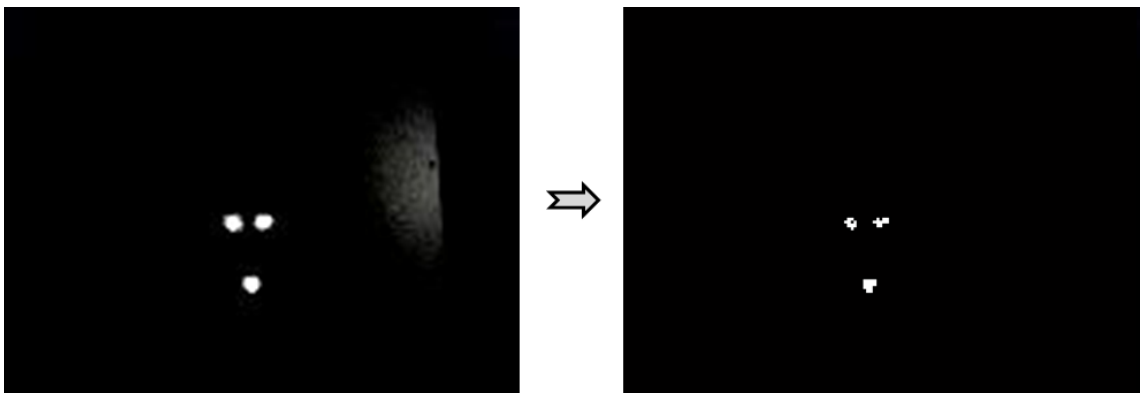

Abbildung 5.30: Binarisierung des erweiterten Bezugspunkts

Durch die binarisierte Aufnahme des weiterentwickelten Bezugspunkts aus der Abbildung 5.30 wird visuell die Pixelvolumina der einzelnen LED's des Musters sowie deren Pixelabstand bestimmt. Daraus geht hervor, dass die Volumina zwischen 5 und 30 Pixel variieren und der Abstand der hinteren LED's des Musters zueinander ca. 8 Pixel und zur Spitze ca. 15 Pixel beträgt. Diese Informationen werden in den übrigen Funktionen benutzt, um die Ausrichtung des Musters zu identifizieren. Im Einzelnen werden dafür die nachfolgenden Schritte durchgeführt:

find_led: In diesem Block werden alle nummerierten Objekte (hier maximal drei) mit den Sollvolumina der LED's verglichen und bei einer Übereinstimmung deren X- und Y-Koordinaten der Mitten an die nächste Funktion weitergegeben.

direction: Damit ein Richtungsvektor aus dem LED Muster interpretiert werden kann, werden hier die Abstände der einzelnen LED's zueinander untersucht. Falls der Abstand größer als 12 Pixel zu den anderen beiden LED's ist, handelt es sich um die Spitze des Richtungsvektors. Der halbe Abstand der anderen beiden LED's zueinander definiert den Anfang des Richtungsvektors. Bei einer falschen Identifizierung des Musters werden die Werte null an die nächste Funktion weitergegeben, die später eine Fehlaufnahme signalisieren.

position: Im letzten Funktionsblock der digitalen Bildverarbeitung wird die Position sowie die Ausrichtung des Roboters berechnet. Als Koordinatenursprung dient dafür die Mitte des vorher bestimmten Richtungsvektors und dessen Richtung bildet die X-Achse. Weiterhin wird der Winkel zwischen diesem Richtungsvektor und der X-Achse des digitalen Bildes bestimmt. Dieser Winkel bildet zugleich die Rotationslage des Robotinos, die in den Regelkreis einfließt. Um die reale Position des Roboters in Abhängigkeit der Rotationslage zu bestimmen, wird die aktuelle Pixelentfernung vom Bildmittelpunkt zum Mittelpunkt des Richtungsvektors um diesen Winkel, in die negative Richtung, mit der Vektorrotation genauso wie in der Simulation gedreht (siehe Abbildung 5.25) und die berechneten Werte in den Regelkreis weitergeleitet. Dieser Zusammenhang wird nochmal in der nachfolgenden Abbildung symbolisch dargestellt:

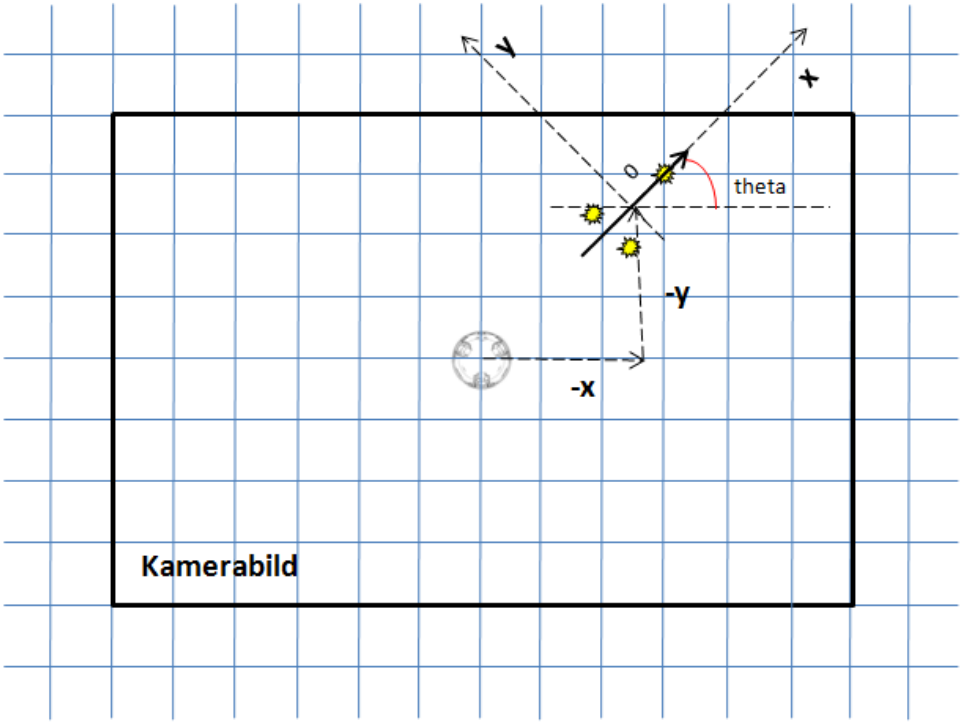

Abbildung 5.31: Positionsbestimmung des Robotinos

Bei einer ungenauen Ermittlung der Mitte der LED's entstehen Winkelfehler von ca. vier Grad. Bezugnehmend auf die Pixelentfernung von 15 Pixel zwischen dem hinteren Teil des weiterentwickelten LED Musters und dem vorderen, entsteht bei einer Rotation von einem Pixel ein Winkelfehler mit dem nachfolgenden Zusammenhang:

$$theta = arctan(\frac{1Pixel}{15Pixel}) = 3,81Grad \tag{18}$$

5.8.3.2 Analyse des realen Reglverhaltens

Zur Analyse des Regelverhaltens wird eine reale Fahrt durchgeführt, die jeweiligen Werte in Matlab importiert und anhand eines Plots mit dem zugehörigen Simulationsmodell verglichen. Die nachfolgende Abbildung zeigt diese Gegenüberstellung:

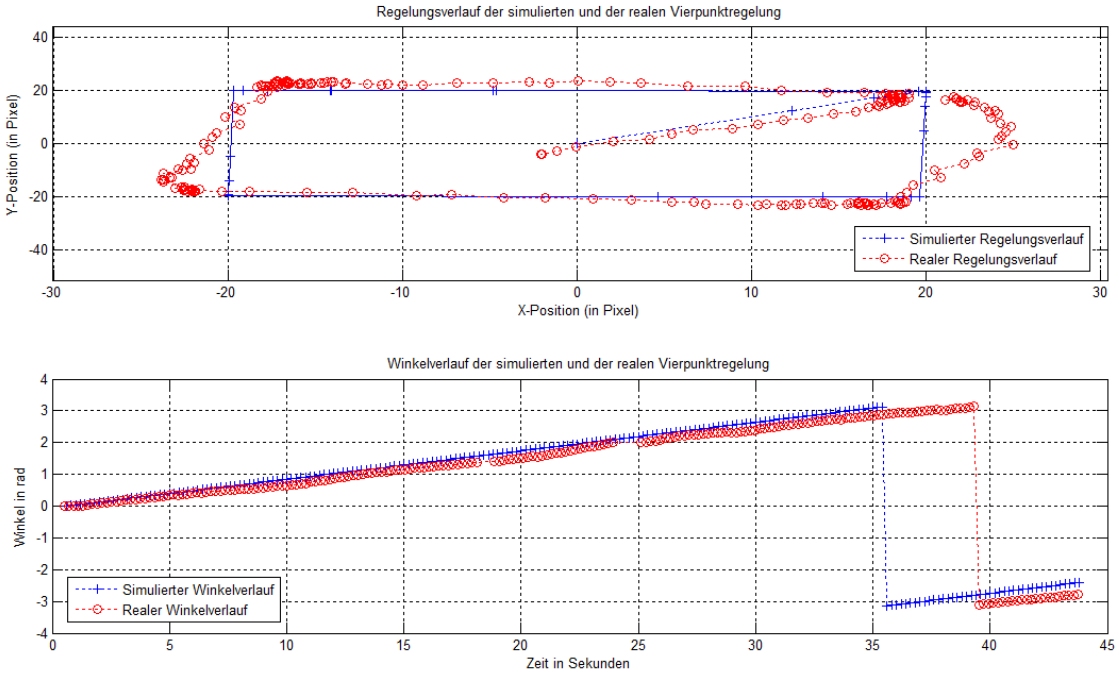

Abbildung 5.32: Vergleich der realen und simulierten Regelverläufe der rotierenden Vierpunktregelung

Im oberen Plot aus der Abbildung 5.32 sind die simulierten und die realen Positionswerte der rotierenden Vierpunktregelung zu sehen. Am Plot ist zu erkennen, dass der reale Regelverlauf die stationären Endwerte erreicht. Bei der realen Fahrt dauert dieser Prozess jedoch aufgrund von Regelabweichungen länger, als bei der Simulation. Durch die Realisierung eines Zielradius von 2 Pixel, entsteht auch bei dem realen Regelverlauf eine etwas schräge Fahrbahn. Wegen einer Schrittzeit von 0,2 Sekunden und einer konstanten Drehbewegung des Robotinos, addiert sich zu dieser Schräge noch eine Regelabweichung hinzu. Im unteren Plot aus der Abbildung 5.32 sind die realen und die simulierten Winkelverläufe zu den zugehörigen Regelfahrten dargestellt. Angesichts der Umrechnungsfaktoren der

Geschwindigkeiten in mm/s sind diese Verläufe beinahe identisch. Die Steigung des realen Winkelverlaufs schwankt etwas im Vergleich zu dem simulierten Werten. Diese Schwankungen entstehen durch Winkelfehler, die anhand der Formel (18) bereits erläutert wurden. Des Weiteren können hier Echtzeitfehler den Winkelverlauf bei der realen Regelfahrt verfälschen. Insgesamt wurde ein gutes Ergebnis erzielt, bei dem die Identifizierung der Robotinoposition sowie der Lage mit dem weiterentwickelten Bezugspunkt möglich ist.

6 Abschluss

In dieser Diplomarbeit wird eine Lösung vorgestellt, wie in der Regelungstechnik zeitgemäße Hardware in Verbindung mit standardisierter Software eingesetzt werden kann, um damit Automatisierungsprozesse zu verbessern. Mit diesem System ist es möglich nahezu jede Computerhardware in die Steuer- und Regelabläufe einzubinden, die mittels Matlab/-Simulink realisiert werden.

6.1 Zusammenfassung

Mit einer USB Webcam wurden in dieser Diplomarbeit am Beispiel der Vorgängermodelle einige Regelprozesse entwickelt, die direkt auf dem System ausgeführt werden und darüberhinaus eine große Auswahl an externen Komponenten unterstützen können. Zu diesem Zweck wurde auf dem Robotino ein Linux Betriebssystem eingerichtet, das die Grundalge für die Treiberentwicklung zur Steuerung des Roboters bildet. Dieses Betriebssystem gilt in der Computertechnik als eine stabile, modulare und frei verfügbare Plattform, die in der Automatisierungstechnik für Embedded Systeme hervorragend geeignet ist. Der weit verbreitete Einsatz dieser Software führt zu einer regelmäßigen Aktualisierung der Kommunikationsschnittstellen, sodass auch künftig innovative Hardware von diesem System unterstützt wird. Die vorgenommenen Konfigurationen des Linux Betriebssystems auf dem Robotino ermöglichen eine autonome Anwendung der Regelmodelle auf einem leistungsschwachen Rechner. Diese Modelle werden auf einem externen Computer realisiert, auf den Robotino übertragen und dort ausgeführt. Unter Verwendung von zusätzlicher Programme und zugehöriger Skripte wurde dieser Übertragungsaufwand auf einige Mausklicks minimiert. Bezugnehmend auf die Effektivität der ganzen Anordnung wurden drei Echtzeit Regelprozesse entwickelt, die den Robotino mit zunehmender Komplexität auf vorgesetzte Zielpositionen regeln. An dieser Stelle wurden zwei unterschiedliche Bezugsobjekte konstruiert, die von der Webcam leicht erkannt werden können und dadurch Informationen über die Koordinationslage beziehungsweise die Ausrichtung des Robotinos liefern. Zur Extraktion dieser Daten aus dem geschossenen Kamerabild werden gängige Funktionen der digitalen Bildverarbeitung verwendet. Die Regelmodelle beinhalten noch einige weitere Schnittstellen zur Ansteuerung der Robotinohardware, jedoch wurden diese nicht benutzt. Zur Untersuchung der realen Regelfahrten wurde zu jedem Modell eine zugehörige Simulation erstellt und deren Verhalten miteinander verglichen. Zu diesem Zweck wurden in eine gesonderte Datei die realen Koordinationsdaten in Abhängigkeit der Zeit während der Fahrt aufgenommen, die im Anschluss in Matlab importiert und anhand von Plots visualisiert werden können. Durch diese Analysen wurde festgestellt, dass die Verläufe sich annähernd gleichen und demzufolge eine zufriedenstellende Regelung des Robotinos mit der USB Webcam realisiert worden ist.

6.2 Fazit

Anhand dieser Arbeit wurde eine Grundlage geschaffen, um mit Matlab/Simulink und einem Linux Betriebssystem beliebige Computerhardware in Steuer- und Regelprozesse einbinden zu können. Ein breites Einsatzgebiet von Linux und dessen Eigenschaften ermöglicht den Anwendern eine zahlreiche Unterstützung von Problemlösungen durch entsprechende Internetforen. Eine eigene Realisierung von Treibern für die Busprotokolle der jeweiligen Schnittstellen ist bei dieser Methode nicht nötig, sodass dadurch Entwicklungszeit und die damit verbundenen Kosten eingespart werden können. Wegen der schwachen Rechenleistung des PC/104 auf dem Robotino sind die realisierten Regelmodelle auf einer minimalen Rechenbasis entwickelt worden. Infolgedessen führt eine Ungenauigkeit von einem Pixel, bei der Identifizierung des Bezugspunkts zu einer relativ großen Positionsabweichung des Robotinos von ca. 1,66 cm. Des Weiteren hat eine Rechenzeit von 0,2 sec des Regelzyklus eine Regelabweichung zufolge, die zusammen mit den Pixelfehlern in den visualisierten Plots im Vergleich zum idealen Verlauf deutlich zu sehen ist. Mit weiterführenden Mitteln der Regelungstechnik und der digitalen Bildverarbeitung könnten diese Abweichungen verkleinert werden. Insgesamt wurde die komplette Zielsetzung erfüllt und ein System entwickelt, das auch in angehenden Projekten angewendet werden kann.

6.3 Ausblick

Bei einer möglichen Erweiterung oder Optimierung dieser Arbeit wäre es sinnvoll, einen Leistungsstärkeren Rechner anstelle des aktuellen PC/104 für die Regelprozesse einzusetzen. Folglich könnte die Rechenzeit für einen Regelzyklus deutlich gesenkt und dadurch eine präzisere Regelung realisiert werden. Weiterhin könnte durch diese Maßnahme eine effizientere Hardware verwendet werden, die die Lokalisierung des Roboters verbessert. Hierbei käme eine Industriekamera mit einem viel größerem Auflösungsvermögen in Frage, die die Bilddaten über einen schnelleren Bus wie Firewire senden kann und gegebenenfalls sogar Lichtwellen im infraroten Bereich wahrnehmen kann. Eine Änderung des LED Lichts vom Muster eine wäre weitere Möglichkeit den Kontrast des Bezugspunkts zu dem Hintergrund zu erhöhen. Der Einsatz von mehreren Bezugspunkten würde den Regelbereich erweitern. Durch die Benutzung der vorhandenen Abstandssensoren könnten zusätzliche Informationen bei den Fahrten eingebunden werden, um beispielsweise Gegenständen ausweichen zu können. Alles in einem wurde mit dieser Diplomarbeit ein Grundstein gelegt, mit dem sich unbegrenzte Möglichkeiten für Weiterentwicklungen öffnen.

6.4 Danksagung

Ich möchte mich hiermit bei Herrn Prof. Dr. Th. Holzhüter für das interessante Thema und für die Möglichkeit zur Realisierung dieser Diplomarbeit bedanken.

Ein besonderer Dank gilt

- Herrn Peter Suchan
- Herrn Dipl. -Ing. Elard Köhler
- Herrn Dipl. -Ing. Simon Wulff
- Herrn Dipl. -Ing. Benjamin Tang
- Herrn Dipl. -Ing. Jens Zeyn-Kranz

die mich immer freundlich unterstützt haben, und mir hilfsbereit und kompetent bei meiner Arbeit zur Seite standen.

Literaturverzeichnis

[About (Zugriff: 17.04.2010)] *Linux / Unix Command: termios*. http://linux.about.com/library/cmd/blcmdl3_termios.htm

[Axelson 2009] Axelson, Jan: *USB Complete*. LAKEVIEW RES, 2009. – ISBN 978–1931448086

[C/C++ Forum (Zugriff: 26.05.2010)] *Linux Webcam*. http://www.c-plusplus.de/forum/viewtopic-var-t-is-249144-and-view-is-previous.html

[Compaq Microsoft National Semiconductor] *OpenHCI*. ftp://ftp.compaq.com/pub/supportinformation/papers/hcir1_0a.pdf. – 1999

[Festo-Didactic (Zugriff: 16.06.2010)] *Festo-Didactic*. http://www.festo-didactic.com/de-de

[HAW (Zugriff: 19.07.2010)] *Hochschule für Angewandte Wissenschaften*. http://de.academic.ru/pictures/dewiki/72/HAW-Hamburg-lot.jpg

[Intel Corporation] *Universal Host Controller Interface (UHCI)*. ftp://ftp.compaq.com/pub/supportinformation/papers/hcir1_0a.pdf. – 1996

[Istockphoto (Zugriff: 19.04.2010)] *Notebook*. http://www.istockphoto.com/file_thumbview_approve/3215941/2/istockphoto_3215941-laptop.jpg

[Kelm 2001] Kelm, Joachim: *USB 2.0*. Franzis, 2001. – ISBN 3–7723–7965–6

[Köhler 2010] Köhler, Elard: *Entwicklung einer kamera-basierten Bahnführungs-Regelung für einen omnidirektionelen Roboter mit Matlab xPC Target*, Hochschule für Angewandte Wissenschaften, Diplomarbeit, 2010

[Kofler 2001] Kofler, Michael: *Linux*. ADDISON-WESLEY, 2001. – ISBN 3–8273–1854–8

[Kontron (Zugriff: 18.06.2010)] *Kontron PC/104*. http://www.kontron.de

[Linux (Zugriff: 1.06.2010)] *Video4Linux API*. http://linux.bytesex.org/v4l2/API.html

[Logitech (Zugriff: 18.06.2010)] *Logitech Quickcam E3500*. http://www.logitech.com/en-us/webcam-communications/webcams

[Mathworks (Zugriff: 16.07.2010)] *Mathworks*. http://www.mathworks.de

[Meiners] Meiners, Prof. Dr.-Ing. U.: *Betriebssysteme*. http://www.etech.haw-hamburg.de. – 2009

[MKSSoftware (Zugriff: 17.04.2010)] *Struct termios*. http://www.mkssoftware.com/docs/man5/struct_termios.5.asp

[Myweb] *Real-Time Workshop 7 Guide*. http://myweb.msoe.edu/~muthuswamy/pubs/ee128Fa09-DOCS/lab3-InvertedPendulumDynamics-INCOMPLETE/RealTimeWorkShopStuff/rtw_ug.pdf. – 2008

[PC-Erfahrungen (Zugriff: 14.04.2010)] *Linux-Befehle*. http://www.pc-erfahrung.de/linux/linux-befehle.html

[PuTTY Download (Zugriff: 23.06.2010)] *PuTTY*. http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

[Quade & Kunst 2006] Quade & Kunst, Jürgen Quade Eva-Katharina K.: *Linux-Treiber entwickeln*. dpunkt, 2006. – ISBN 3–89864–392–1

[Robotino] *Robotino Handbuch*. http://www.festo-didactic.com/ov3/media/customers/1100/544305_robotino_deen.pdf. – 2007

[Schimek] Schimek, Michael H.: *Video for Linux Two API Specification*. http://v4l2spec.bytesex.org/v4l2spec/v4l2.pdf. – 2008

[Senne 1992] Senne, Achim: *UNIX-Lexikon*. IWT, 1992. – ISBN 3–88322–372–7

[Techbus (Zugriff: 20.04.2010)] *RS-232*. http://techpubs.sgi.com/library/tpl/cgi-bin/getdoc.cgi/hdwr/bks/SGI_EndUser/books/PChall_L_OG/sgi_html/ch02.html

[Ubuntu (Zugriff: 17.03.2010)] *Ubuntu*. http://www.ubuntu.com

[Weber 1993] Weber, Wolfgang: *DOS Batch-Programmierung*. Harry Deutsch, 1993. – ISBN 3–8171–1266–1

[Wikibook (Zugriff: 17.04.2010)] *Serial Programming/termios*. http://en.wikibooks.org/wiki/Serial_Programming/termios

[Wikipedia (Zugriff: 7.04.2010)] *Universal Serial Bus*. http://de.wikipedia.org/wiki/Universal_Serial_Bus

[WinSCP (Zugriff: 29.06.2010)] *WinSCP*. http://www.winscp.net

Abkürzungsverzeichnis

AIN ..... Analog Input
AP ..... Access Point
BIOS ..... Basic Imput Output System
CF ..... Compact Flash
DDR ..... Double Data Rate
DI ..... Digital Input
DO ..... Digital Output
EHCI ..... Enhanced Host Controller Interface
ERT ..... Embedded Real Time
GND ..... Ground
HAW ..... Hochschule für Angewandte Wissenschaften
I/O ..... Input/Output
IP ..... Internet Protocol
IR ..... Infrarotsonsor
ISA ..... Industry Standard Architecture
JPEG ..... Joint Photographic Experts Group
LAN ..... Local Area Network
LED ..... Light Emitting Diode
NRZI ..... Non Return to Zero
OHCI ..... Open Host Controller Interface
PCI ..... Peripheral Component Interconnect
PPM ..... Portable PixMap
RD ..... Receive Data
REL ..... Relais
RTW ..... Real Time Workshop
SW ..... Schwarzweiß
TD ..... Transmit Data
UHCI ..... Universal Host Controller Interface
USB ..... Universal Serial Bus
VGA ..... Video Graphics Array
WLAN ..... Wireless Local Area Network

Anhang (Hinweis)

Der Anhang befindet sich komplett auf der dieser Diplomarbeit zugehörigen CD bei Herrn Prof. Dr. Thomas Holzhüter.

Erklärung über die selbstständige Bearbeitung des Themas

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.

Hamburg, 24. September 2010	
Ort, Datum	Unterschrift