FH Vorarlberg
University of Applied Sciences

Embedded Systems 2

Fachhochschule Vorarlberg
Master Mechatronics

Betreut von

Prof. (FH) Dipl.-Ing. Horatiu Pilsan

Vorgelegt von

Florian Burtscher
Roman Passler

Dornbirn, 29.06.2017

Inhaltsverzeichnis

Abbildungsverzeichnis **III**

Tabellenverzeichnis **IV**

Listings **V**

1 Projektbeschreibung **1**
1.1 Laboraufbau . . . 1
1.2 Local-Mode . . . 1
1.3 Chain-Mode . . . 2
1.4 Motorregelung . . . 3

2 Anforderungen **4**
2.1 Use-Case Diagramm . . . 4
2.2 Sequenz Diagramm . . . 5
2.3 Activity Diagramm . . . 6
2.4 Funktionale und nicht Anforderungen . . . 7
2.5 Regelgüte . . . 11

3 Statische Software Struktur **12**
3.1 Designmodell . . . 12
3.2 Implementierungsmodell . . . 13

4 Verhaltensmodell **17**
4.1 Designmodelle . . . 17
4.2 Implementierungsmodelle . . . 21

5 Reglerimplementierung und Verifizierung **26**
5.1 Reglermodell . . . 26
5.2 Reglerkoppelung mit der Statemachine . . . 27
5.3 Verifizierung . . . 27

Abkürzungsverzeichnis **32**

Literaturverzeichnis **33**

Anhang **34**

A Display **35**

B Tastenfeldbelegung **36**

C Kommunikation zwischen den Förderbändern und Master **37**

D Telnet Kommunikation **38**

E Kalman Filter **39**

F Code der Statemachine **40**

Abbildungsverzeichnis

1.1 Geschwindigkeitsprofil 2
1.2 Förderbandkette 3

2.1 Use-Case Diagramm 5
2.2 Initialisierung 6
2.3 Aktivitätsdiagramm 7
2.4 Definition Sprungantwort 11

3.1 Designmodell-Klassendiagramm 13
3.2 Implementiertes Klassendiagramm 15

4.1 Designmodell-Statemachine Choose Operating Mode 17
4.2 Designmodell-Substatemachine Locale Mode 18
4.3 Designmodell-Substatemachine Chain Mode 19
4.4 Designmodell-Subsubstatemachine Drive Profile 20
4.5 Implementierte-Statemachine Choose Operating Mode 21
4.6 Implementierte-Substatemachine Locale Mode 22
4.7 Implementierte-Substatemachine Chain Mode 23
4.8 Implementierte-Subsubstatemachine Performing 24
4.9 Implementierte-Subsubstatemachine Drive Profile 25

5.1 Motormodell 26
5.2 Closed Loop Stepresponse mit Reglerparametern vom Auftraggeber . 28
5.3 Closed Loop Stepresponse mit eigenen Reglerparametern 29
5.4 Rampe Soll Ist Verlgeich 30
5.5 Rampe Soll Ist Verlgeich mit schnellerer Beschleunigung und Verzögerung 31

Tabellenverzeichnis

1.1 Werte des Geschwindigkeitsprofils 2

2.1 Allgemeine funktionale Anforderungen 8
2.2 Funktionale und nicht Anforderungen für Local-Mode 9
2.3 Funktionale und nicht Anforderungen für Chain-Mode 10

3.1 Beschreibung der Tasks 16

5.1 Regler Parameter 28

Listings

F.1 Statemachine mit den Membermethoden 40

1 Projektbeschreibung

In diesem Projekt ist Herr Prof. (FH) Dipl.-Ing. Horatiu Pilsan der Auftraggeber. Der Auftraggeber stellt als Aufgabe, eine Software für ein virtuelles Förderband mit dem vorhanden Laboraufbau zu erstellen. Der DC-Motor fungiert hierbei als Simulation des Förderbandantriebes. Die Förderbänder können entweder lokal betrieben (Local-Mode) oder zusammengeschlossen werden, um eine geschlossene Kette einzurichten (Chain-Mode). Der Zweck ist, virtuelle Pakete mit dieser Kette zu transportieren. Es gibt im gesamten fünf virtuelle Förderbänder, die als Ring zusammengeschlossen werden können.

1.1 Laboraufbau

Für die Simulation wird ein Laboraufbau verwendet, der ein Förderband praxisnah simuliert. Dieser Aufbau beinhaltet die folgenden Komponenten:

- Mikrocontrollerboard mit VxWorks v6.9 als Betriebssystem
- DC-Motor als Förderbandantrieb
- Display und Keyboard für das User Interface
- 2 Netzwerkverbindung

In den folgenden Abschnitten erfolgt eine kurze Beschreibung der zu realisierenden Betriebsarten Local-Mode und Chain-Mode.

1.2 Local-Mode

Der Local-Mode, der Service-Betriebsmodus, ist für Situationen vorgesehen, wo ein manuelles Eingreifen benötigt wird. Es muss möglich sein, das Profil wie in Abbildung 1.1 dargestellt in beide Richtungen zu starten. Vor dem Start des Profils kann die Geschwindigkeit in einem definierten Bereich geändert werden. Die Bedienung des Förderbandes erfolgt über die Tastatur oder über eine Teletype Network (Telnet) Verbindung.

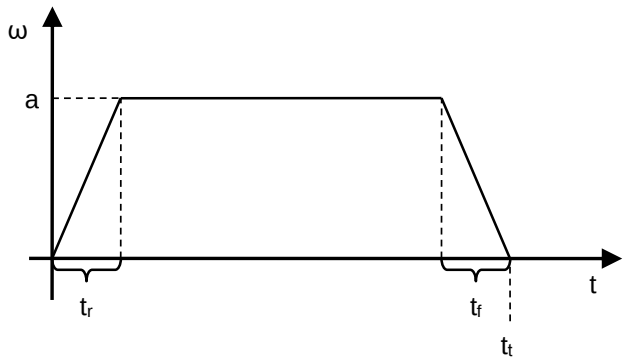

Abbildung 1.1: Geschwindigkeitsprofil
Quelle: Pilsan 2017

In Tabelle 1.1 sind die geforderten Werte vom Auftraggeber für das Geschwindigkeitsprofil aus Abbildung 1.1 dargestellt.

Bezeichnung	**Wert**	**Kommentar**
ω	$1000 \dots 2200\ U/min$	Drehzahl
a	$1800\ U/min$	Amplitude
t_r	$1\ s$	Anstiegszeit
t_f	t_r	Abfallzeit = Anstiegszeit
t_t	$8\ s$	Gesamtzeit

Tabelle 1.1: Werte des Geschwindigkeitsprofils
Quelle: Pilsan 2017

1.3 Chain-Mode

Der Chain-Mode beschreibt den vollautomatischen Betriebsmodus, in welchem die Pakete vom linken Förderband zum rechten Förderband weitergereicht werden. Um Pakete in die virtuelle Förderbandkette einzubringen, wird ein Masterförderband verwendet, welches bestimmt, wo ein Paket in die Kette eingebracht wird. In Abbildung 1.2 ist die Netzwerkstruktur der virtuellen Förderbandkette dargestellt. Vorgesehen sind insgesamt 5 Förderbänder, es können jedoch $n \in (1, 20)$ Förderbänder hinzugefügt werden.

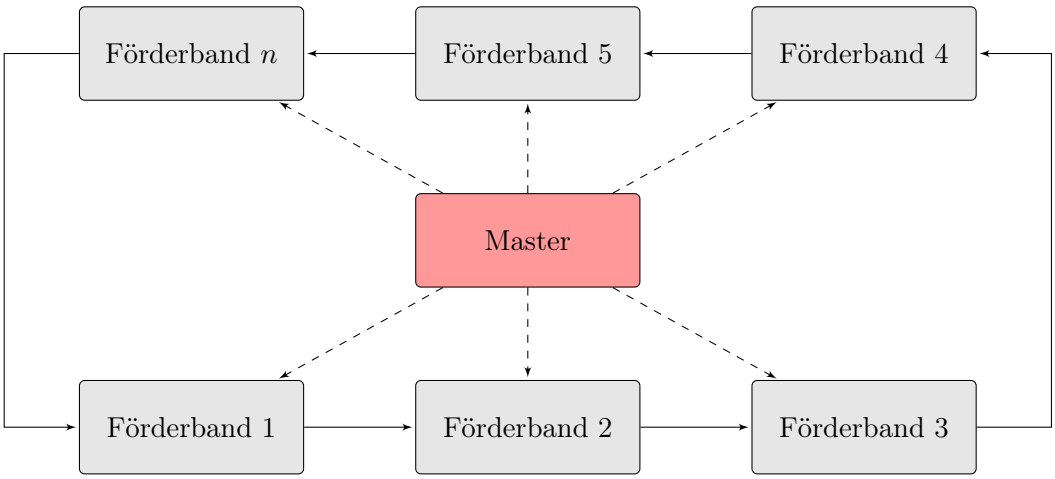

Abbildung 1.2: Förderbandkette
Quelle: eigene Ausarbeitung

Das Masterförderband ist nicht Teil der Förderbandkette. Befindet sich ein Paket auf dem virtuellen Förderband, so wird dieses von links nach rechts weitertransportiert und dem rechten Förderband übergeben. Mittels eines Handshakes wird die Paketübergabe und die Paketabnahme kontrolliert und gesteuert.

1.4 Motorregelung

Die Geschwindigkeit des Motors muss in einer geschlossenen Schleife gesteuert werden, der Code für die Steuerung wird zur Verfügung gestellt.

2 Anforderungen

In diesem Kapitel wird die allgemeine Funktion mithilfe von Use-Case Diagrammen, Sequenz Diagrammen und Activity Diagrammen beschrieben. Anschließend werden die funktionale Anforderung (FR) und nicht funktionale Anforderung (NFR) in Tabellenform kurz und prägnant dargestellt.

2.1 Use-Case Diagramm

In Abbildung 2.1 ist der Use-Case für das Conveyor Belt (CB) dargestellt. Es gibt im gesamten vier Akteure:

User Dieser Akteur ist für den Service-Betriebsmodus vorgesehen und führt an einem Förderband Operationen im Local-Mode durch. Der User erhält volle Kontrolle über alle möglichen Funktionen des Förderbandes. Durchgeführt wird die Kommunikation entweder über die lokale Tastatur oder über eine Telnet-Verbindung.

Master Dieser Akteur ist nur im Chain-Mode vorhanden. Er vermittelt den Förderbändern die jeweils rechten Nachbarn und übergibt einem der Förderbänder in der Kette ein Paket und initiiert somit den Start der Förderbänder. Diese Kommunikation erfolgt über Standard Ethernet.

left CB Dieser Akteur ist nur im Chain-Mode vorhanden. Er dient dazu, die Kommunikationsanforderungen zwischen dem linken und dem eignen Förderband darzustellen. Diese Kommunikation erfolgt über Standard Ethernet.

right CB Dieser Akteur ist nur im Chain-Mode vorhanden. Er dient dazu, die Kommunikationsanforderungen zwischen dem eigenen und dem rechten Förderband darzustellen. Diese Kommunikation erfolgt über Standard Ethernet.

Es gibt im gesamten 4 Use-Cases die die Funktionalität des zu entwerfenden Systems beschreiben:

Operating Mode Hier erfolgt die Auswahl zwischen Local-Mode und Chain-Mode

Local Mode Betrieb im Local-Mode

Chain Mode / Packet transport Betrieb im Chain-Mode

Set right CB IP Dient zur Initialisierung der Förderkette und weist die jeweiligen rechten Nachbarn zu.

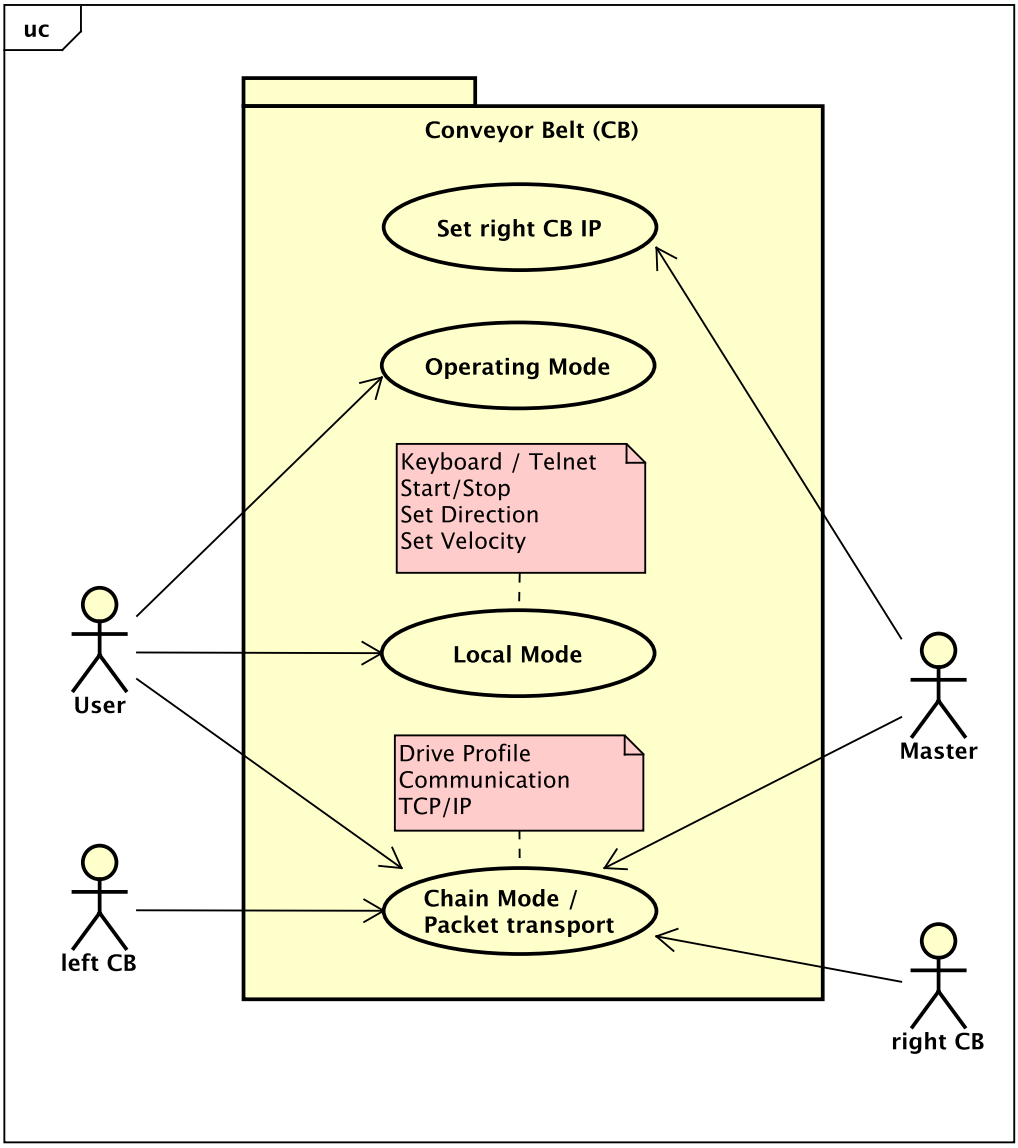

Abbildung 2.1: Use-Case Diagramm
Quelle: eigene Ausarbeitung

2.2 Sequenz Diagramm

In Abbildung 2.2 ist das Sequenzdiagramm dargestellt. Es zeigt die Initialisierung der Förderbandkette durch den Master. Dieser übergibt nach der Vermittlung der jeweiligen rechten Nachbarn das zu transportierende Paket eines Förderbandes, die daraufhin den Ablauf zum Fördern starten. Als Master im Chain-Mode ist ein beliebiger Rechner der FHV gedacht.

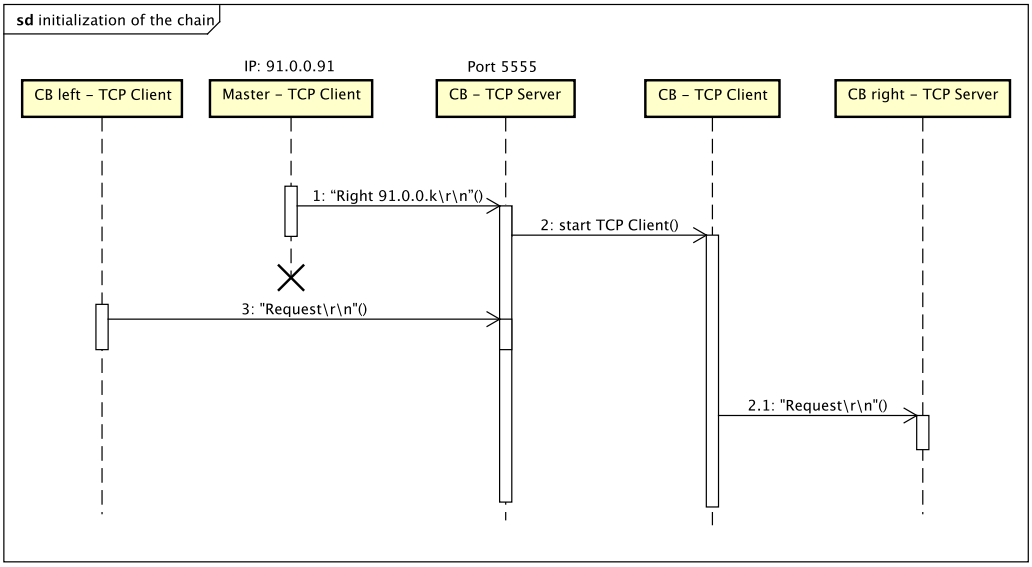

Abbildung 2.2: Initialisierung
Quelle: eigene Ausarbeitung

2.3 Activity Diagramm

In Abbildung 2.3 ist das Aktivitätsdiagramm dargestellt. Es beschreibt den Local-Mode und Chain-Mode.

Im Chain-Mode muss das virtuelle Förderband nur Pakete von links nach rechts befördern. Es soll gewartet werden, bis von links eine Anfrage gesendet wird, dass ein Paket vorhanden ist. Wird gerade eine Bewegung ausgeführt, so muss mit „WAIT“ geantwortet werden. Andernfalls wird mit „READY“ geantwortet und für eine Zeitdauer von $t_{pp} = 1\ s$ mit einer langsamen Geschwindigkeit von 100 U/min in Richtung zum rechten Förderband gefahren. Nach Ablauf der Zeitdauer t_{pp} wird das linke Förderband mit „RELEASE“ informiert, dass das Paket erfolgreich übernommen wurde. Ebenfalls wird jetzt das definierte Geschwindigkeitsprofil aktiviert.

Ist das Geschwindigkeitsprofil beendet, wird eine „REQUEST“ Anfrage zum rechten Förderband gesendet, um festzustellen, ob dieses bereit ist. Wird die Anfrage mit „WAIT“ quittiert, so wird das Förderband gestoppt. Jedoch bei Quittierung mit „READY“, wird die Geschwindigkeit auf 100 U/min reduziert, bis „RELEASE“ empfangen wurde. Jetzt kehrt das Förderband wieder in den Ruhezustand zurück und stoppt den Motor.

Der Local-Mode unterscheidet sich im wesentlichen zum Chain-Mode, dass die Förderbänder autark arbeiten. Der Local-Mode ist somit unabhängig und kein Teil der Förderkette und dadurch entfällt jegliche Kommunikation zur Außenwelt, bis auf die Telnet-Verbindung zur Steuerung.

In Local-Mode kann zuerst die Richtung der Förderbewegung und die Drehzahl im Bereich von $1000 \ldots 2200$ U/min in Schritten von 100 U/min eingestellt werden. Die Benutzerinteraktion erfolgt durch das Keyboard und Display oder Telnet.

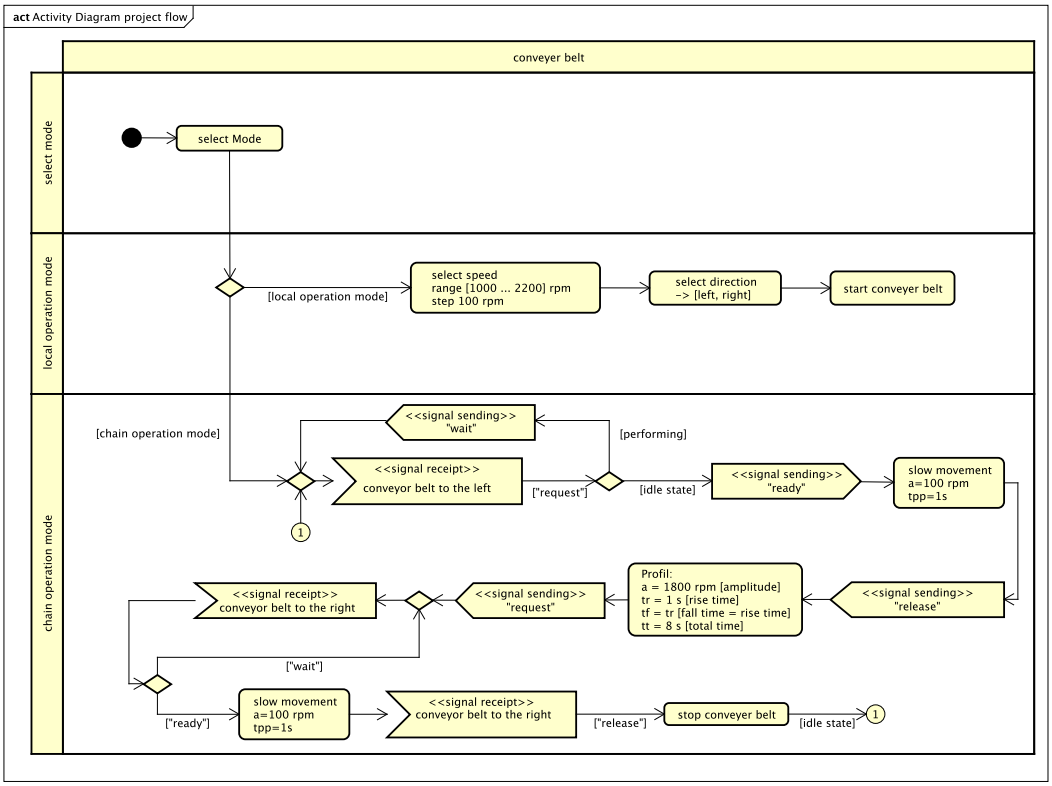

Abbildung 2.3: Aktivitätsdiagramm
Quelle: eigene Ausarbeitung

2.4 Funktionale und nicht Anforderungen

Die funktionalen Anforderungen wurden zur besseren Übersicht in drei Teile gegliedert:

- Allgemeine funktionale Anforderungen
- Funktionale Anforderungen für Local-Mode
- Funktionale Anforderungen für Chain-Mode

Nr.	FR[a] NFR[b]	Verbindlichkeit	Beschreibung
1	FR	muss	Das Förderband muss zwei Betriebsmodi unterstützen. Den Local-Mode und Chain-Mode.
2	FR	muss	Das System muss hingehend erweiterbar sein, sodass Förderbänder beliebig hinzugefügt bzw. entfernt werden können.
3	FR	muss	Das Förderband muss einen PTP-Client implementiert haben, um die Uhrzeit zum Master zu synchronisieren. Der Code wird vom Auftraggeber zur Verfügung gestellt.
4	NFR	muss	Das PTP muss Port 5432 verwenden.
5	FR	muss	Die Geschwindigkeit muss über eine geschlossene Reglerschleife geregelt sein. Der benötigte Code wird vom Auftraggeber zur Verfügung gestellt.
6	FR	soll	Die in Pilsan 2014 definierten Funktionen sollten für den Zugriff der Hardware verwendet werden. In diesem Dokument werden die notwendigen Funktionen für den Zugriff der Hardware beschrieben.
7	FR	muss	Das Keyboard muss als Eingabe fungieren. Die Tastenbelegung ist unter Tabelle B.1 ersichtlich.
8	FR	muss	Telnet muss als Eingabe fungieren. Die Telegramme sind unter Tabelle D.1 ersichtlich.
9	NFR	muss	Der Telnet-Server muss den Port 4444 verwenden.
10	NFR	muss	a und ω müssen $\pm 50\ U/min$ genau erreicht werden.
11	NFR	muss	Das Förderband muss folgende Informationen am Display ausgeben (siehe Abbildung A.1) • Angewählter Betriebsmodus
12	NFR	soll	Die PTP Implementierung kann vom Auftraggeber übernommen werden.
13	NFR	muss	Die PTP Updaterate („Sync“ Message) ist auf 4 Sekunden gestellt und verwendet nicht Multicast.
14	NFR	muss	Das Feld „Nanoseconds“ des PTP ist immer null.

Tabelle 2.1: Allgemeine funktionale Anforderungen
Quelle: eigene Ausarbeitung

[a] funktionale Anforderung (FR)
[b] nicht funktionale Anforderung (NFR)

Nr.	FR[a] NFR[b]	Verbindlichkeit	Beschreibung
15	FR	muss	Das Förderband muss unabhängig von den anderen Förderbändern gesteuert werden können.
16	FR	muss	Das Förderband muss in beide Richtungen (links und rechts) das Geschwindigkeitsprofil abfahren können. Ein Tippbetrieb ist ausgeschlossen.
17	FR	muss	Das Förderband muss fähig sein, das Geschwindigkeitsprofil jederzeit zu unterbrechen.
18	FR	muss	Die Ansteuerung muss bei Stillstand erfolgen.
19	NFR	muss	Die Geschwindigkeit muss zwischen $1000 \ldots 2200\ U/min$ eingestellt werden können.
20	NFR	muss	Die Geschwindigkeit muss in Abständen von $100\ U/min$ verändert werden können.
21	NFR	muss	Das Förderband muss folgende Informationen am Display ausgeben (siehe Abbildung A.1): • eingestellte maximale Profildrehzahl • angewählte Richtung

Tabelle 2.2: Funktionale und nicht Anforderungen für Local-Mode
Quelle: eigene Ausarbeitung

[a] funktionale Anforderung (FR)
[b] nicht funktionale Anforderung (NFR)

Die Anforderung Nummer 19 hat sich während der Entwicklungszeit auf Wunsch des Auftraggebers von $100 \ldots 2200\ U/min$ auf $1000 \ldots 2200\ U/min$ geändert.

Nr.	FR[a] NFR[b]	Verbindlichkeit	Beschreibung
22	FR	muss	Das Förderband fördert nur Pakete vom linken zum rechten Förderband.
23	FR	muss	Das Förderband muss auf Übergabeanforderungen vom linken Förderband reagieren und darauf antworten.
24	FR	muss	Befindet sich bereits ein Paket auf dem Förderband und ein „REQUEST“ trifft vom linken Förderband ein, so muss mit einem „WAIT“ geantwortet werden.
25	FR	muss	Ist das Förderband leer, so muss auf eine „REQUEST“ Anfrage vom linken Förderband mit „READY“ geantwortet werden.
26	FR	muss	Nach der Antwort „READY“ muss dass Förderband die Geschwindigkeit auf 100 U/min reduzieren und diese Geschwindigkeit für eine Zeit von $t_{pp} = 1\ s$ beibehalten.
27	FR	muss	Sobald die Zeitdauer t_{pp} abgelaufen ist, muss dem linken Förderband mit „RELEASE“ mitgeteilt werden, dass das Paket erfolgreich übernommen wurde.
28	FR	muss	Das Förderband muss nach der erfolgreichen Übernahme das Geschwindigkeitsprofil (Abbildung 1.1) aktivieren.
29	FR	muss	Nach vollendetem Geschwindigkeitsprofil muss der Motor gestoppt werden.
30	FR	muss	Nach dem Stoppen des Motors muss das Förderband dem rechten Förderband eine „REQUEST“ Anfrage senden.
31	FR	muss	Lautet die Antwort des rechten Förderbandes „WAIT“, wird der Motor gestoppt.
32	FR	muss	Lautet die Antwort des rechten Förderbandes „READY“, muss der Motor mit 100 U/min gestartet werden, bis das rechte Förderband mit „RELEASE“ quittiert.
33	NFR	muss	Das Förderband muss folgende Informationen am Display ausgeben (siehe Abbildung A.1): • Status des Förderbandes („WAIT“, „RELEASE“, „READY“, „REQUEST“, Profil abfahren). • Status der Initialisierung der Kette. • Sende- und Empfangszeitpunkt des Paketes.
34	FR	muss	Für die Kommunikation zum linken Förderband bzw. Master muss ein Transmission Control Protocol (TCP) Server zur Verfügung stehen. Zum rechten Förderband muss ein Client die Verbindung aufnehmen. Die Telegramme sind unter Tabelle C.1 ersichtlich.
35	NFR	muss	Für den TCP Server und Client muss Port 5555 verwendet werden.

Tabelle 2.3: Funktionale und nicht Anforderungen für Chain-Mode
Quelle: eigene Ausarbeitung

[a] funktionale Anforderung (FR)
[b] nicht funktionale Anforderung (NFR)

2.5 Regelgüte

In Abbildung 2.4 sind die Parameter einer Sprungantwort dargestellt. Für das Projekt wird ein PI-Regler im Simulink bzw. Matlab zur Verfügung gestellt, für den wie folgt die Regelgüte definiert wird. Die Regelgröße ist im Beharrungszustand, wenn die Regelgröße im definierten Toleranzbereich von $\pm$ 50 rpm liegt. Die Anregelzeit T_{an} soll schneller als 30 ms erfolgen und die Ausregelzeit T_{aus} soll 120 ms nicht übersteigen. Das Überschwingen x_{ue} darf nicht größer als 30 % sein.

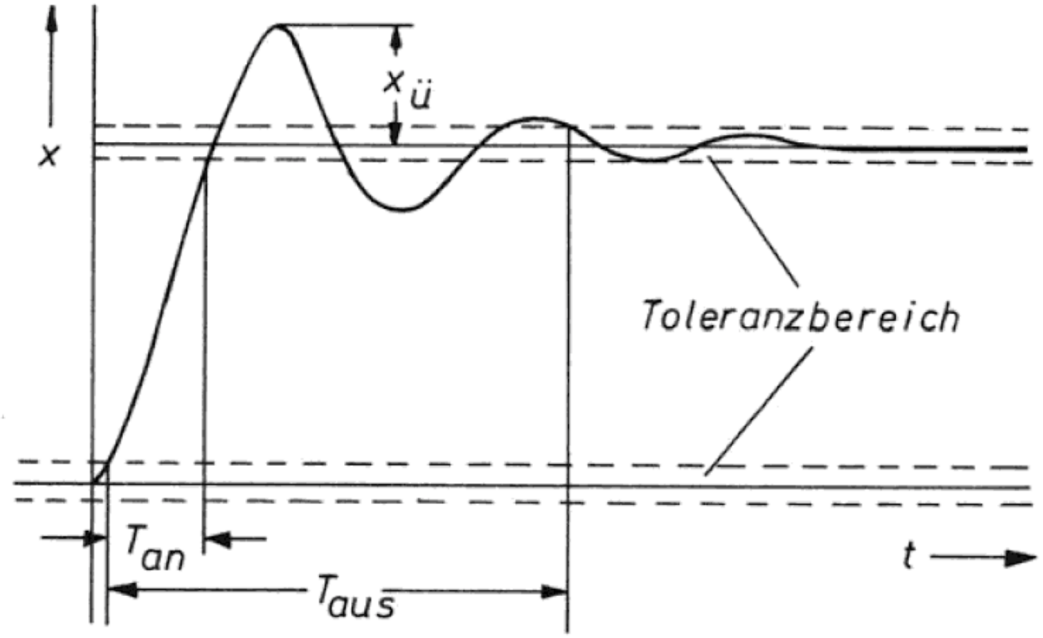

Abbildung 2.4: Definition Sprungantwort
Quelle: Berend Brouër 1998

3 Statische Software Struktur

In diesem Kapitel wird das statische Softwarekonzept vorgestellt. Für die Erstellung wird Unified Modeling Language (UML) verwendet und mittels Astah[1] erstellt.

3.1 Designmodell

In Abbildung 3.1 ist die erstellte statische Software Struktur abgebildet. Zentrale Komponente der Architektur stellt die Klasse „ConveyorBelt“ dar. Diese erstellt und verwaltet mittels Statemachine sämtliche für die Anwendung benötigten Objekte.

Alle aktiven Klassen, inklusive der Klasse Display, werden als eigenständige Tasks ausgeführt. Die Klassen „TCPServer“, „TCPClient“, „TelnetServer“ und „Keyboard“ dienen zur Kommunikation zwischen Förderbändern sowie zwischen Förderband und menschlichem Operator. Das Interface „ICommunication“ stellt sicher, dass Klassen mit bidirektionaler Kommunikation die grundlegenden Methoden implementiert haben müssen.

Die Klasse „Controller“ beinhaltet den Regler für den Förderbandmotor und die Logik für das Abfahren der Geschwindigkeitsprofile. Der Controller erstellt darüber hinaus ein Objekt vom Typ „IMotor“, welches mit der Klasse „Motor“ initialisiert wird.

Die Klasse „Display“ dient zur Visualisierung der aktuellen Zustandsgrößen. Es verwendet die „Getter“ Funktionen der Klassen „Controller“ und „ConveyorBelt“.

[1] `http://astah.net`

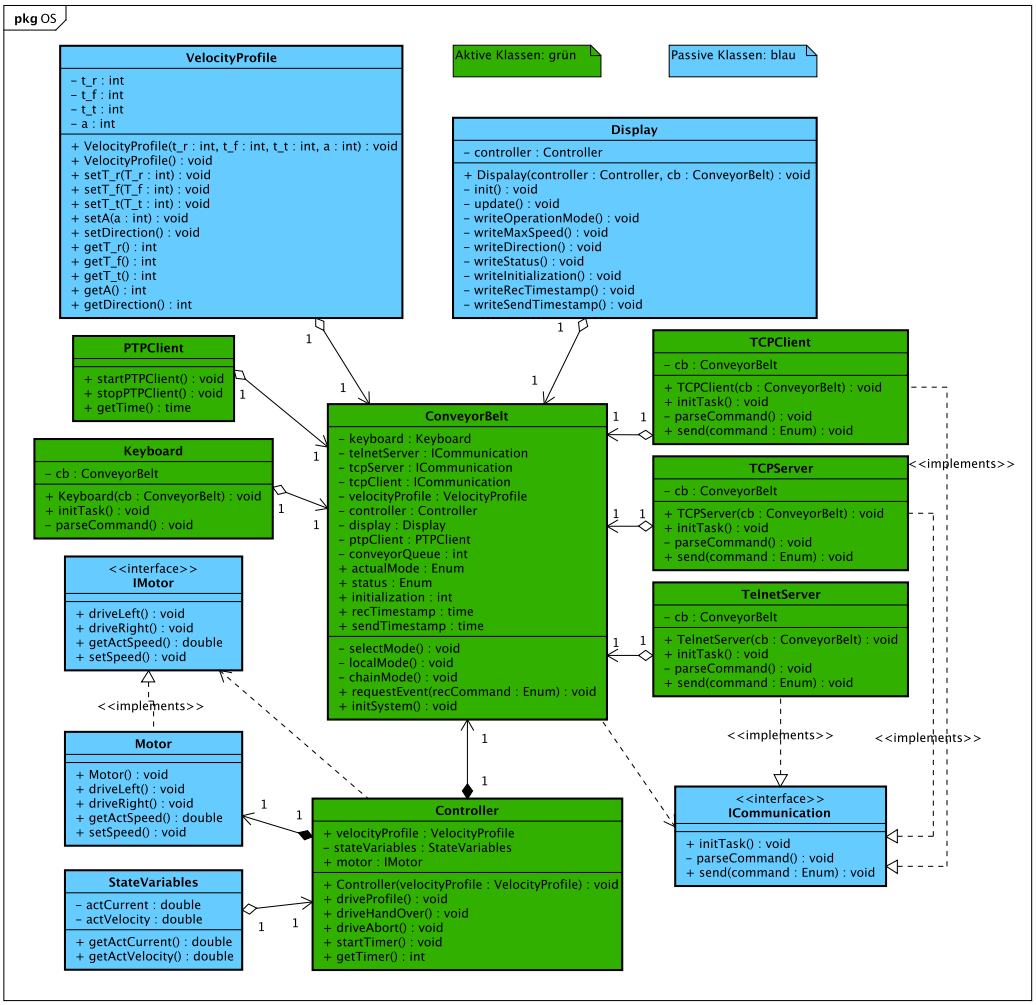

Abbildung 3.1: Designmodell-Klassendiagramm
Quelle: Eigene Ausarbeitung

3.2 Implementierungsmodell

In diesem Abschnitt werden die Unterschiede und wichtige Implementierungsdetails im Vergleich zum Designmodell erläutert. In Abbildung 3.2 ist das implementierte Klassendiagramm dargestellt. Im Vergleich zum Designmodell in Abbildung 3.1 wird die Klasse „ConveyorBelt“ jetzt „SystemManager“ genannt.

Weiters wird der „Parser“ jetzt über eine eigene Klasse implementiert. Das bedeutet die Kommunikationsschnittstellen erstellen ein Objekt von der Klasse „Parser“ über welches sie bei Empfang einer Eingabe über die Methode „parseCommand()“ zugreifen. Dabei stellt der „Parser“ fest, ob das Kommando für die Schnittstelle zulässig ist und gibt einen „Enum“ mit dem zulässigen Kommando zurück. Die jeweilige Kommunikationsschnittstelle reicht in weiterer Folge über eine Instanz von der Klasse „SystemManager“ über den Methodenaufruf „requestEvent“ den erhaltenen „Enum“

weiter. Je nach „Enum“ wird dann ein Event der Klasse „StateMachine“ gesendet, welches das Verhaltensmodell von Abschnitt 4.2 implementiert.

Ein weiters Implementierungsdetail betrifft die Vermischung von C und C++ Code in Klassen die Tasks spawnen. Dabei wird über eine „Task init“ Methode der Task gespawnt. Es wird ein „WorkerTask“ von der Klasse erstellt, in dem von der eigenen Klasse ein Objekt erstellt und die „run“ Methode ausgeführt wird. Wird diese Vorgehensweise implementiert, können Instanzen von anderen C++ Klassen im „WorkerTask“ verwendet werden, ansonsten müsste über globale Variablen der Informationsaustausch erfolgen, dadurch könnten nicht autorisierte Klassen auf Variablen zugreifen, das wird mit diesem Modell verhindert.

Im Designmodell ist eine Klasse für den „TCPClient“ und eine für den „TelnetServer“ vorgesehen. Da beide Klassen den selben Typ von Schnittstelle implementieren, wird im Implementierungsmodell für diese jeweils eine Instanz von der Klasse „TcpServer“ erstellt. Das kann auch dadurch realisiert werden, weil der „Parser“ im Implementierungsmodell in einer eigenen Klasse ausgelagert wird und dieser je nach Instanz zwischen TCPServer und TelnetServer Verbindung unterscheidet und die zulässigen Kommandos als „Enum“ liefert.

Das Interface „IMotor“ und die Klasse „StateVariables“ wurden im Implementierungsmodell nicht umgesetzt.

Die Klasse „Controller“ beinhaltet das Reglermodell, welches mit Hilfe von Simulink modelliert und exportiert wurde. Das Modell wurde in C++ Code adaptiert. Die Taktfrequenz wird mit einem Softwarewatchdog mit jener des SystemManagers synchronisiert. Das Verhalten des Geschwindigkeitsprofils vom Motor ist ebenfalls in dieser Klasse implementiert.

Weiters wurde die Klasse „UdpClient“ hinzugefügt. Diese wird für den Nachrichtenaustausch zwischen PTP-Client und PTP-Server verwendet.

Um die Drehzahl zu ermitteln, wurde eine „Encoder“ Klasse erstellt. Diese implementiert jeweils zwei unterschiedliche Arten der Drehzahlmessung.

Counter Zeitdifferenz Hier wird die Zeitdifferenz der Änderung zum letzten Encoderimpuls ermittelt. Diese Implementierung unterstützt die Ausgabe von Rohdaten, gemittelten Daten und mittels Kalman 1D gefilterte Daten.

Counter gemittelte Werte Hier werden die Encoderompulse gezählt und über die verstrichene Zeit vom letzten Aufruf der Methode, kann die Drehzahl ermittelt werden.

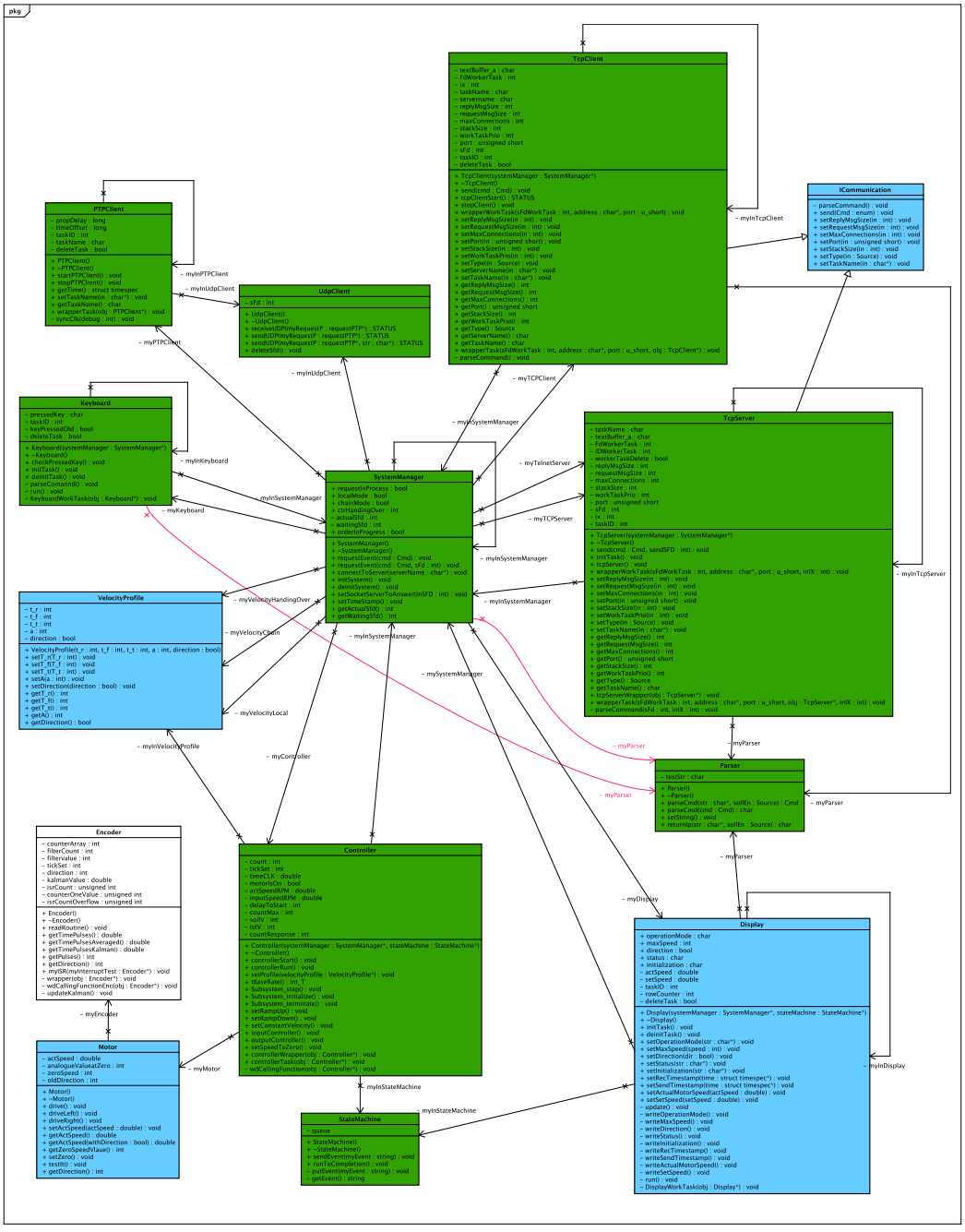

Abbildung 3.2: Implementiertes Klassendiagramm
Quelle: Eigene Ausarbeitung

In Tabelle 3.1 sind die Tasks, die von der Applikation gestartet werden, kurz beschrieben.

	Priorität	Beschreibung
tMain	100	Dieser Task initialisiert alle Klassen und bleibt in der „blocking“ Funktion der Statemachine stehen, d.h. er ist für die Abarbeitung der Events der Statemachine zuständig.
tBaseRate	50	Der Task wird für den Regler benötigt und hat eine zyklische Periode von 15 ms.
tController	150	Ist der Task, der für den Regler benötigt wird, um ihn wieder zu beenden.
tDisplay	110	Dieser Task beschreibt zyklisch alle 50 ms das Display neu.
tEncoder	90	Dieser Task ist für die Auswertung des Kalmanfilters zuständig und hat eine Periode von 5 ms.
tKeyboard	110	Dieser Task ist für die Eingabe der Tastatur am Versuchsaufbau zuständig und arbeitet mit einer Periode von 80 ms.
tPtpClient	110	Dieser Task ist für den Empfang und das Senden der Precision Time Protocol (PTP) Pakete zuständig und läuft azyklisch, da „blocking“ Funktionen verwendet werden.
tTcpServer	101	Dieser Task ist für den Verbindungsaufbau zu den Förderbändern und den Master zuständig und läuft azyklisch, da „blocking“ Funktionen verwendet werden.
FdWorkerTask**x**	101	Dieser Task empfängt die Daten und läuft azyklisch, da „blocking“ Funktionen verwendet werden. Das „**x**“ steht für einen Workertask von 1 bis 10.
tTcpClient	107	Dieser Task empfängt die Daten und läuft azyklisch, da „blocking“ Funktionen verwendet werden.
tTcpTelnet	102	Dieser Task empfängt die Daten und läuft azyklisch, da „blocking“ Funktionen verwendet werden.

Tabelle 3.1: Beschreibung der Tasks
Quelle: Eigene Ausarbeitung

4 Verhaltensmodell

In diesem Kapitel wird das Verhaltensmodell vorgestellt.

4.1 Designmodelle

Folgende Zustandsdiagramme beschreiben das Verhalten des Förderbandes im Local-Mode und Chain-Mode. Der Wechsel zwischen Local-Mode und Chain-Mode ist nur im „Idle-State“ erlaubt.

Bevor das Förderband im Chain-Mode zum Pakettransport benutzt werden kann, bedarf es einer Betriebsmodiauswahl. Diese ist in Abbildung 4.1 dargestellt.

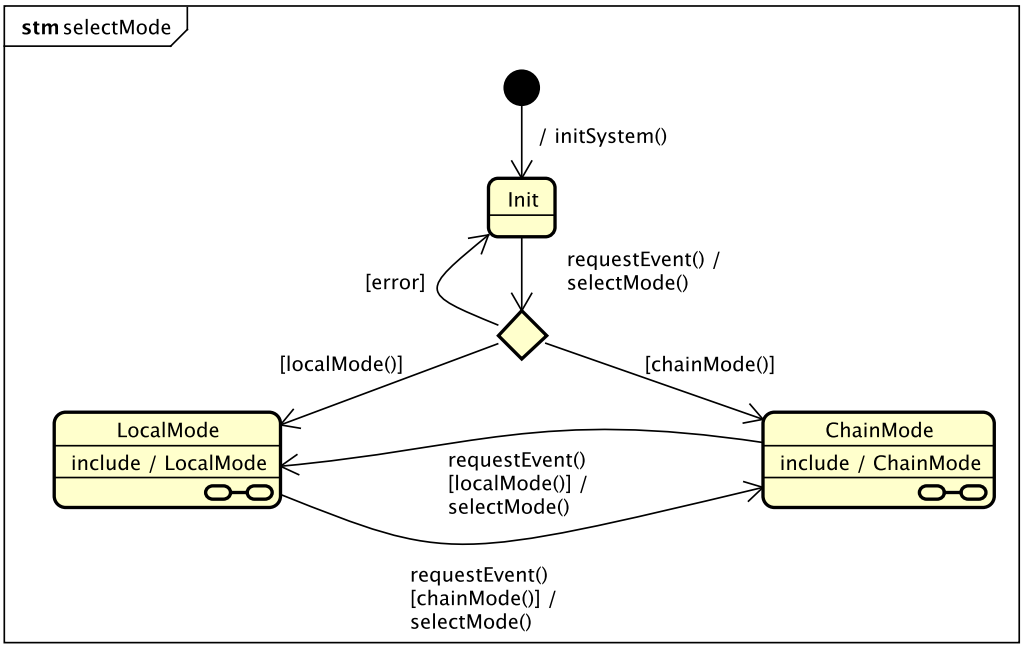

Abbildung 4.1: Designmodell-Statemachine Choose Operating Mode
Quelle: Eigene Ausarbeitung

Ist der Betriebsmodus Local-Mode angewählt, so wird die Statemachine in Abbildung 4.2 ausgeführt.

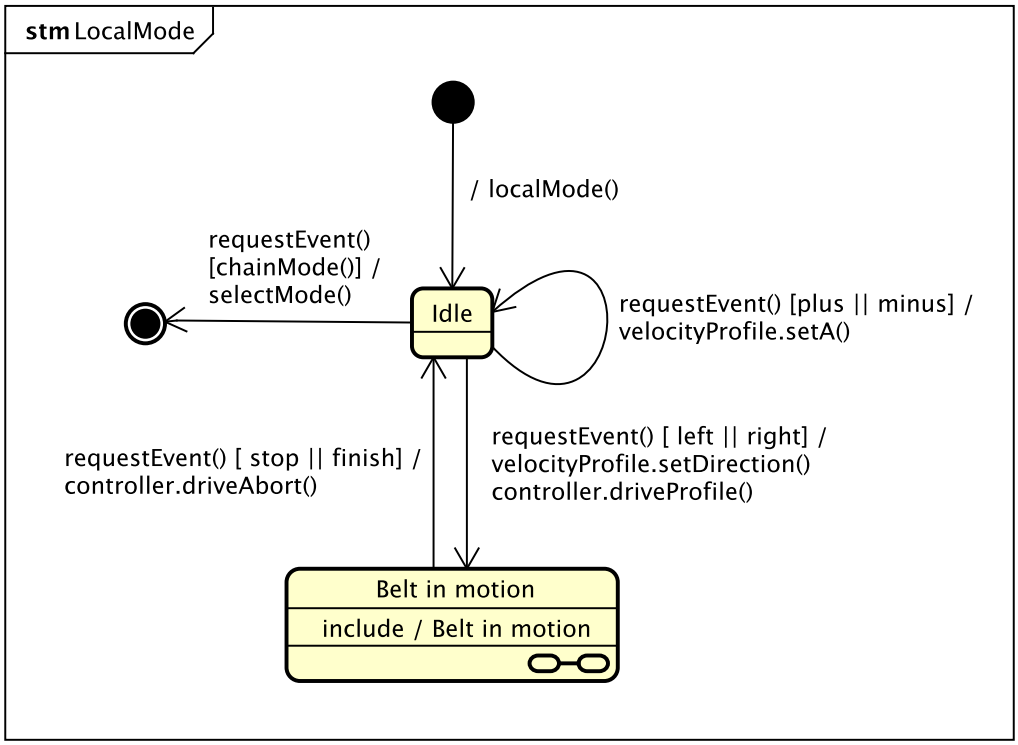

Abbildung 4.2: Designmodell-Substatemachine Locale Mode
Quelle: Eigene Ausarbeitung

Hingegen ist der Betriebsmodus Chain-Mode angewählt, so wird die Statemachine in Abbildung 4.3 ausgeführt.

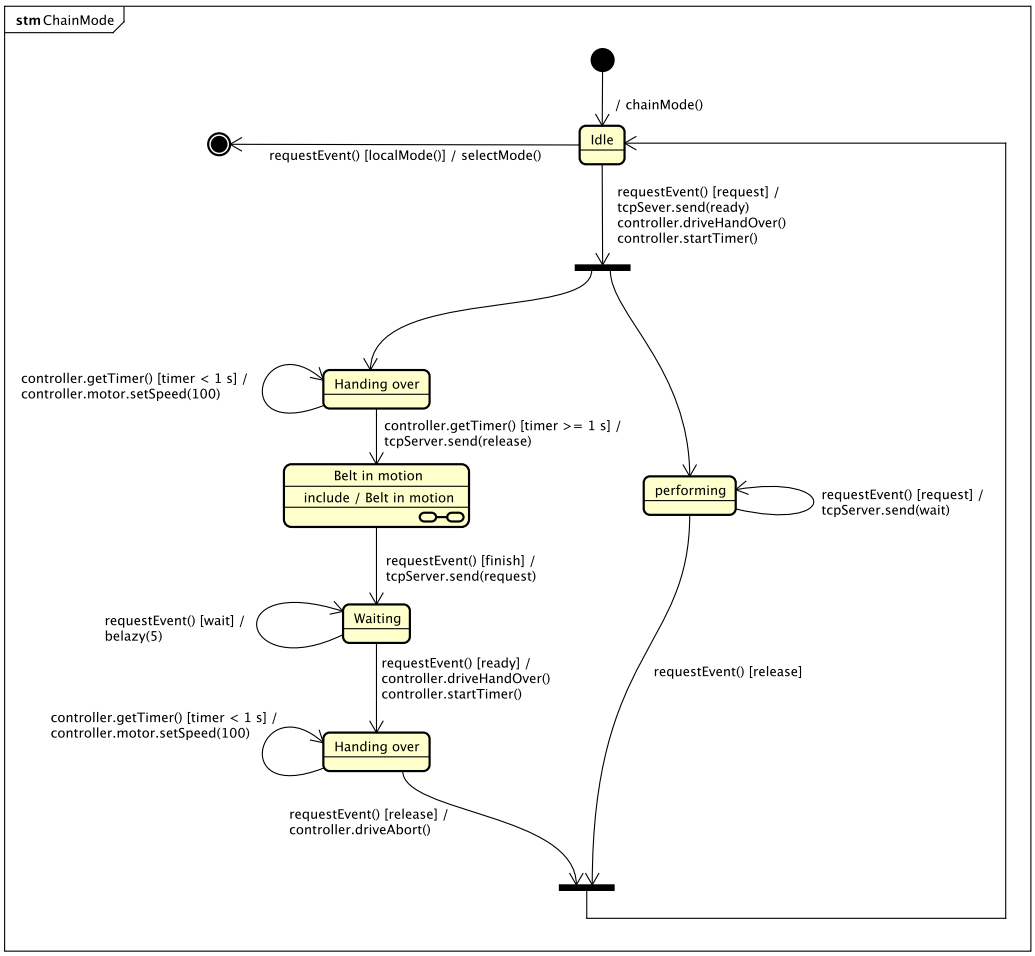

Abbildung 4.3: Designmodell-Substatemachine Chain Mode
Quelle: Eigene Ausarbeitung

Die Statemachines von Local-Mode und Chain-Mode in den Abbildungen 4.2 und 4.3 verwenden für den Zustand „Belt in motion“ die Subsubstatemachine in Abbildung 4.4.

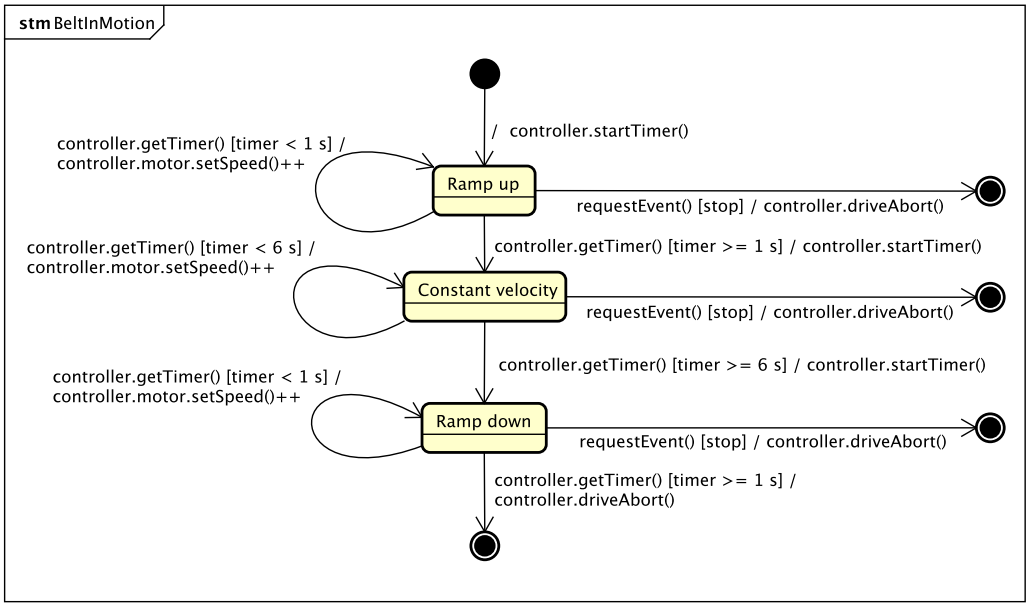

Abbildung 4.4: Designmodell-Subsubstatemachine Drive Profile
Quelle: Eigene Ausarbeitung

4.2 Implementierungsmodelle

In diesem Abschnitt wird die implementierte Statemachine erläutert. Dabei werden die Unterschiede zum Designmodell hervorgehoben. In den folgenden Diagrammen enthalten sind die Trigger (Events) und die Bedingungen, welche eine Transition zwischen zwei Zuständen auslöst sowie die Methode, die dabei aufgerufen wird, die in Anhang F als Codeausschnitt eingesehen werden können. In Abbildung 4.5 ist die Statemachine für die Betriebsmodiauswahl dargestellt. Im Vergleich zum Designmodell wurde der Zustand „startSystem“ hinzugefügt.

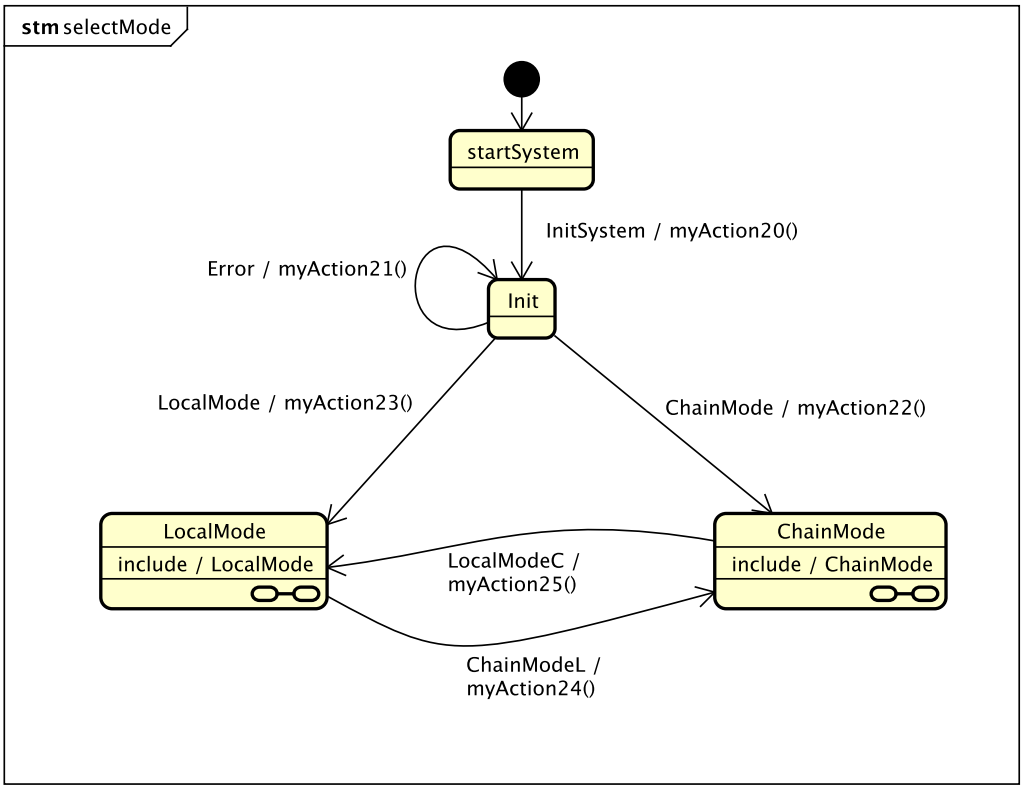

Abbildung 4.5: Implementierte-Statemachine Choose Operating Mode
Quelle: Eigene Ausarbeitung

Das Verhalten vom Local-Mode ist in Abbildung 4.6 dargestellt. Für die Initialisierung der Substatemachine wurde der Zustand „SelectLocalMode“ hinzugefügt. Wird der Local-Mode Betriebsmodi gewählt, ändert sich der Zustand auf „Idle“. Im „Idle“ Zustand kann wiederum auf den Chin-Mode gewechselt werden. Für das Erhöhen und Reduzieren der Maximalgeschwindigkeit wurde eine eigene Transition implementiert. Das selbe gilt für das Starten des Profils über die Richtungsauswahl links oder rechts.

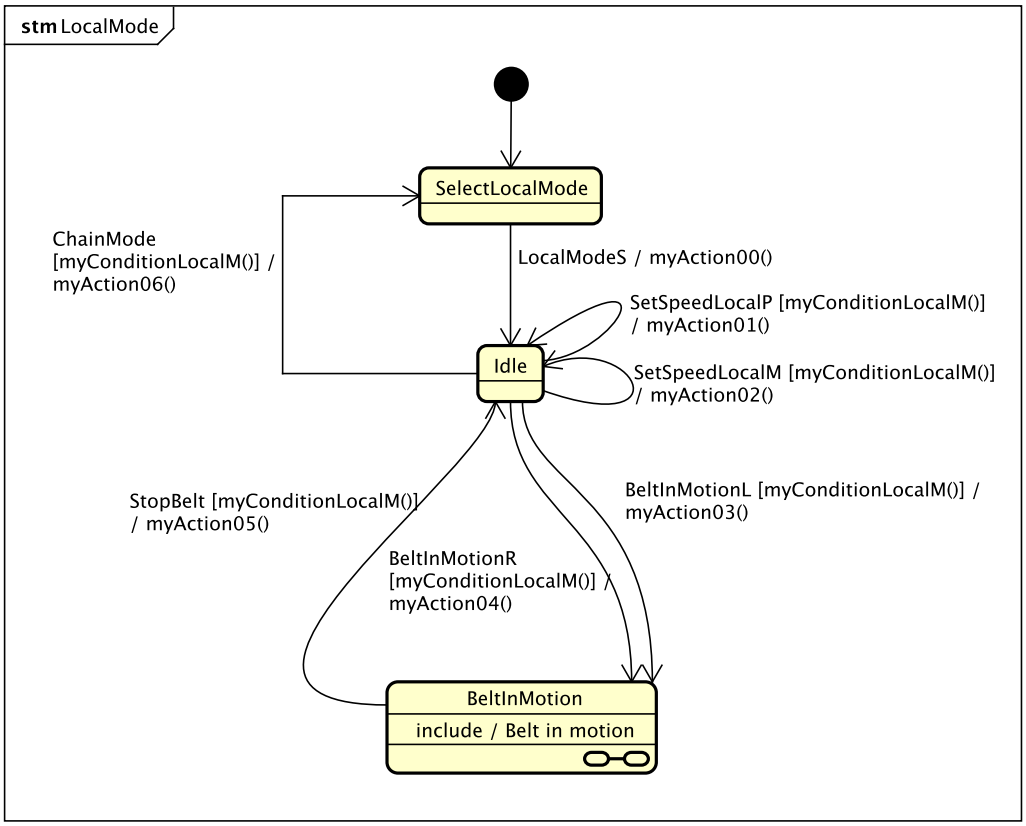

Abbildung 4.6: Implementierte-Substatemachine Locale Mode
Quelle: Eigene Ausarbeitung

In Abbildung 4.7 ist die implementierte Substatemachine, welche das Verhalten vom ChainMode repräsentiert, dargestellt. Im Vergleich zum Designmodell wurde der Zustand „SelectChainMode“ für die Initialisierung hinzugefügt, der Zustand „Performing“ durch eine Subsubstatemachine ersetzt und beim Zustand „Waiting“ eine Abbruch-Transition hinzugefügt. Tritt aus unbekannten Gründen ein Fehler in der Paketübermittlung auf, kann mit dem „stop“ Kommando bzw. mit der Keyboard Eingabe „c“ der Chain-Mode resetet werden.

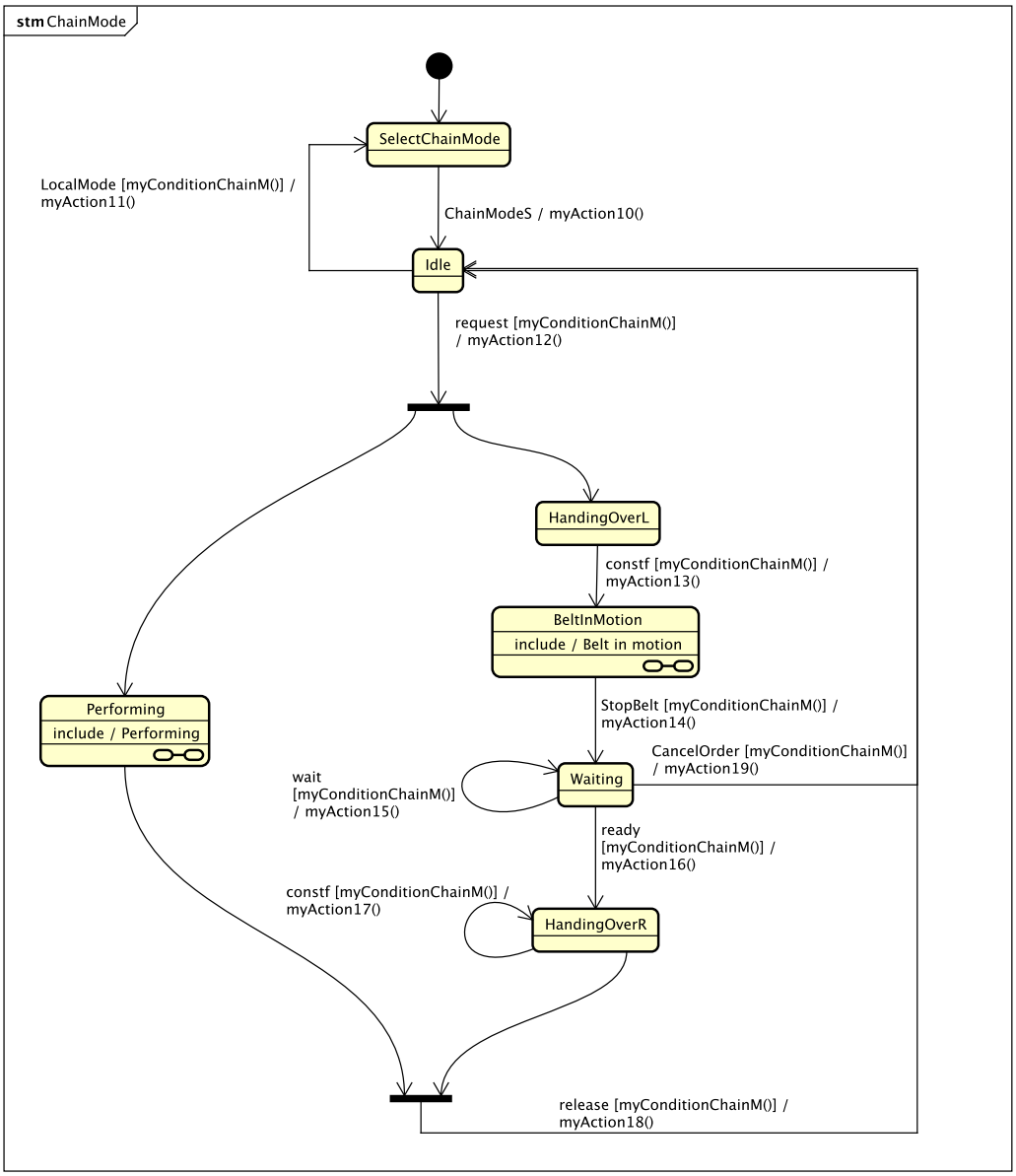

Abbildung 4.7: Implementierte-Substatemachine Chain Mode
Quelle: Eigene Ausarbeitung

Die Abbildung 4.8 zeigt die neu hinzugefügte Subsubstatemachine „Performing“. Wird gerade ein Paket transportiert und erhält das Conveyor Belt einen „Request“ wird die Methode „myAction41()“ ausgeführt. Die Anfrage wird in einem Puffer gespeichert und sobald sich der „ChainMode“ im Zustand „Idle“ befindet, behandelt.

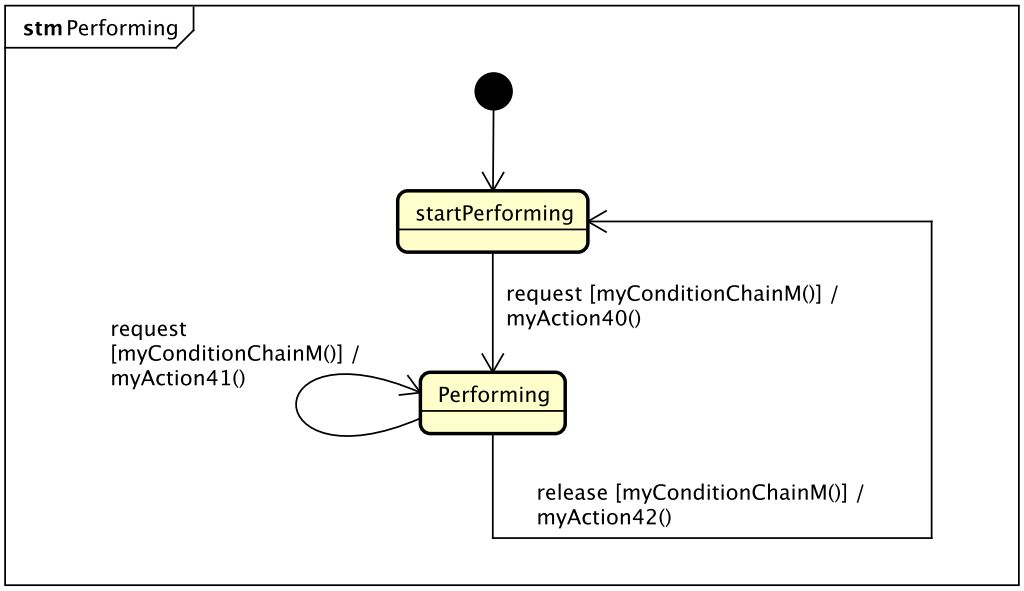

Abbildung 4.8: Implementierte-Subsubstatemachine Performing
Quelle: Eigene Ausarbeitung

Das Abfahren des Geschwindigkeitsprofils ist in Abbildung 4.9 dargestellt. Im Implementierungsmodell ist kein Timer wie im Designmodell enthalten. Dieser wird jetzt in der Klasse „Controller“ implementiert.

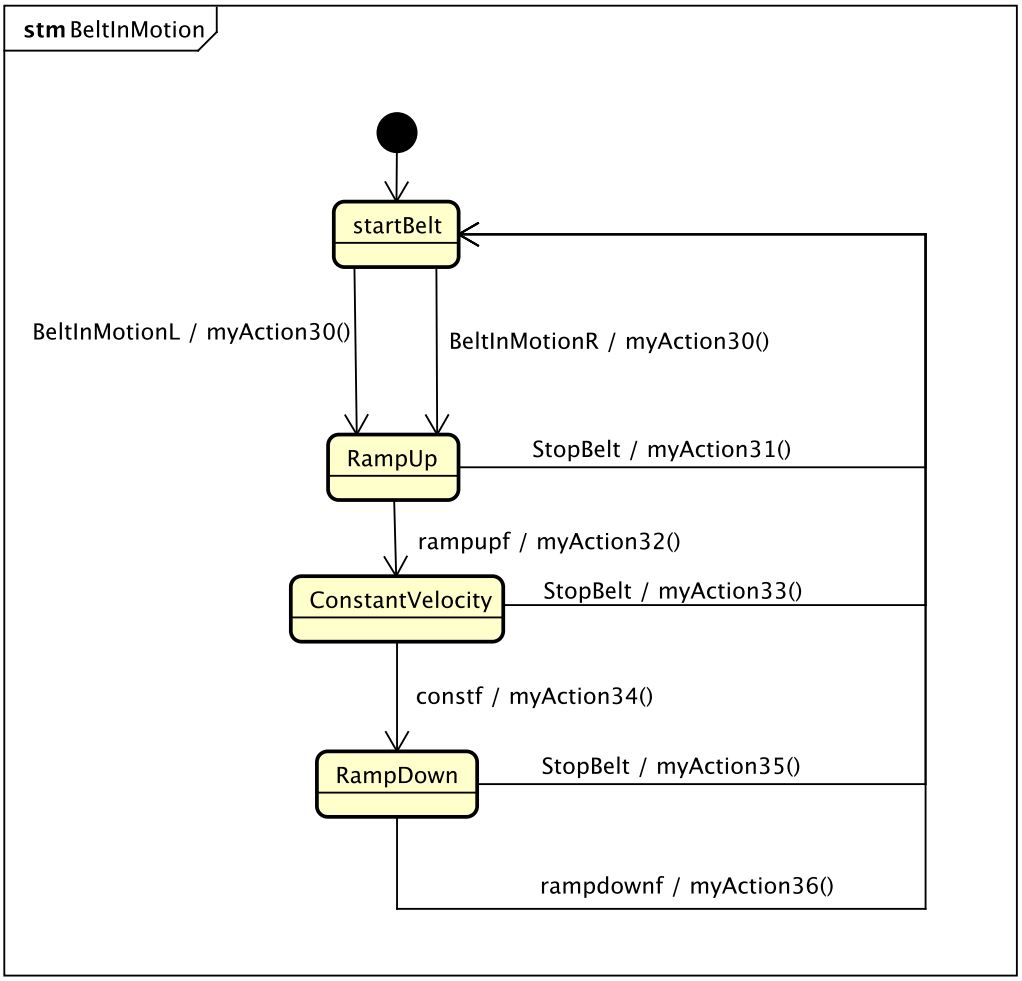

Abbildung 4.9: Implementierte-Subsubstatemachine Drive Profile
Quelle: Eigene Ausarbeitung

5 Reglerimplementierung und Verifizierung

5.1 Reglermodell

In Abbildung 5.1 ist das Modell des implementierten Motors und des PI-Reglers dargestellt. Das Subsystem ist hierbei der PI-Regler, der als diskreter Regler ausgeführt wurde. Das Subsystem wird mittels Simulink in C-Code übersetzt, der direkt für VxWorks verwendet werden kann. Der Eingangsparameter ist die Istgeschwindigkeit und der Ausgangsparameter die Stellgröße (Motorspannung). Jedoch wurde der generierte C-Code von uns auf C++ portiert und der Zeitgeber von einer asynchron laufenden Clock auf einen Softwarewatchdog programmiert. Dies hat den Vorteil, dass das Anwenderprogramm und der Controller synchron gekoppelt sind.

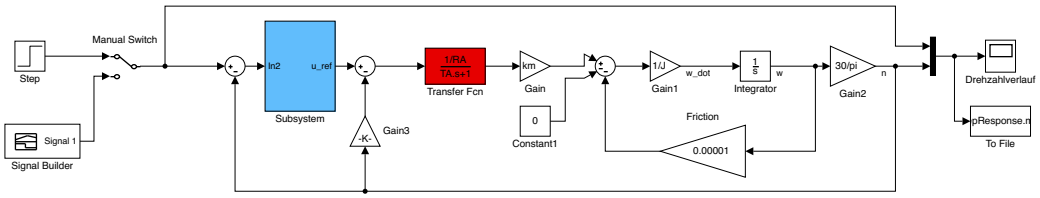

Abbildung 5.1: Motormodell
Quelle: Eigene Ausarbeitung

Die Drehzahl als Eingangsgröße wird von der Klasse Motor via Membermethode „getActSpeed()“ zur Verfügung gestellt. Die Klasse „Motor“ ruft wiederum eine Membermethode der Klasse „Encoder“ auf. Die „Encoder“ Klasse implementiert die Logik für die Auswertung der Drehzahl. Gleichung (5.1) gibt die implementierte Logik an. Der Faktor 10^6 kommt von der Auflösung des Zählers. Dieser zählt in μ-Schritten, aber die Winkelgeschwindigkeit wird in Sekunden benötigt.

$$\omega = \frac{1}{\text{Auflösung Encoder}} \cdot 60\ s \cdot 10^6 \frac{1}{x} \tag{5.1}$$

$$\omega = \frac{1.0}{64.0} \cdot 60.0 \cdot 10^6 \cdot \frac{1}{x} \tag{5.2}$$

$$x \ldots \text{gefilterter Wert} \tag{5.3}$$

Der gefilterte Wert wird von der Klasse „Kalman“ zur Verfügung gestellt. Für die Implementierung des Kalman Filters sei auf Anhang E verwiesen.

Mit Gleichung (5.4) wird die Sollgröße, die vom PI-Regler berechnet wurde, auf den DC-Wandler umgerechnet.

$$u_{out} = \frac{\text{Auflösung}}{2.0} \cdot \frac{1}{\text{max. Ausgangsspannug}} \cdot U_{soll} \tag{5.4}$$

$$u_{out} = \frac{4095.0}{2.0} \cdot \frac{1}{12.0\ V} \cdot U_{soll} \tag{5.5}$$

$$U_{soll} \ldots \text{vom Regler berechnete Ausgangspannung} \tag{5.6}$$

Der Wert u_{out} wird mittels der Membermethode „setActSpeed()“ und „drive()“ von der „Motor“ Klasse übergeben und diese ist dann zuständig, den gewünschten Wert auf den Ausgang zu schreiben.

5.2 Reglerkoppelung mit der Statemachine

Der Regler ist eine eigenständige Klasse, die folgende Membermethoden zur Verfügung stellt, um den Verlauf der Rampe zu steuern.

- „setProfile()“:Setzt die Beschleunigung und Verzögerungswerte und maximale Geschwindigkeit.
- „setRampUp()“: Es soll bis zur maximalen Geschwindigkeit beschleunigt werden.
- „setRampDown()“: Es soll bis zum Stillstand verzögert werden.
- „setConstantVelocity()“: Es soll eine konstante Geschwindigkeit gefahren werden.
- „setSpeedToZero()“: Stopp der Bewegung.

Die Statemachine kann auf diese Membermethoden zugreifen und je nach Zustand die Methoden ausführen.

Der Controller meldet seinen Zustand über die von der Statemachine zur Verfügung gestellte Methode „sendEvent()“ zurück.

5.3 Verifizierung

In Abbildung 5.2 ist die Sprungantwort mit den Regelparametern des Auftraggebers abgebildet. Sie enthält den Vorgabewert („Step“), die simulierte („Stepresponse Simulink“) und zwei gemessene Sprungantworten („Stepresponse gemessen“, „Stepresponse gemessen neu“). Die blaue Kurve ist die Sprungantwort, die ohne Kalmanfilter und mit den gemittelten Zählerwerten aufgenommen wurde. Es ist ersichtlich, dass die Auflösung sehr schlecht ist. Dagegen stimmt die violette Kurve, die mit dem Kalmanfilter und der Zeitdifferenz des Zählers aufgenommen wurde, mit der Wirklichkeit sehr gut überein. Die Risetime T_s und die Ausregelzeiten T_r sind bei beiden nahezu dieselben und liegen weit über den in Abschnitt 2.5 definierten Werten.

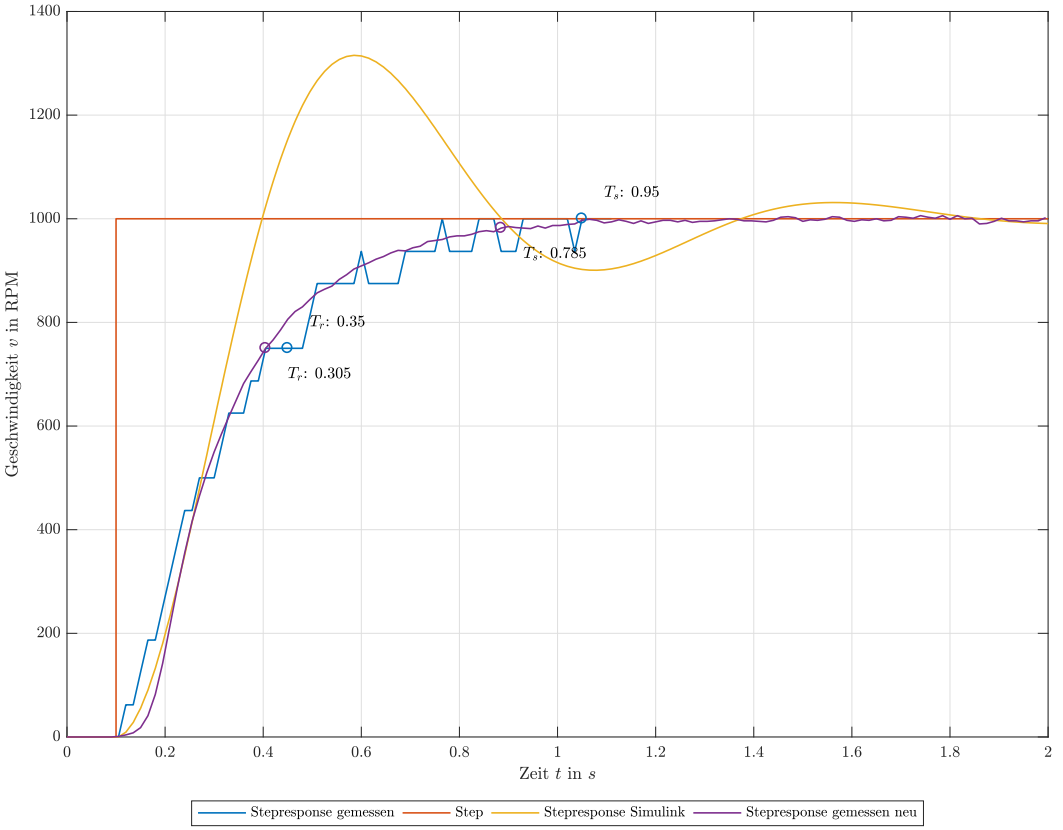

Abbildung 5.2: Closed Loop Stepresponse mit Reglerparametern vom Auftraggeber
Quelle: Eigene Ausarbeitung

Aus diesem Grund wurden eigene Regelparameter verwendet, die in Abbildung 5.3 dargestellt sind. In Tabelle 5.1 sind die verwendeten Reglerparameter festgehalten.

$K_{p,n}$ gibt die Verstärkung des P-Reglers und $K_{i,n}$ die des I-Reglers an. $t_{s,n}$ ist die Zykluszeit, mit der der diskrete Regler arbeitet.

	Standarwerte	Neue Werte
$K_{p,n}$	$8.0 \cdot 10^{-6}$	0.00048
$K_{i,n}$	0.015625	0.0351
$t_{s,n}$	0.015625	0.015

Tabelle 5.1: Regler Parameter
Quelle: Eigene Ausarbeitung

Es ist zu erkennen, dass sich die Risetime T_r und ebenfalls die Ausregelzeit T_s halbiert haben. Das Überschwingen beträgt ca. 2 %. Es können jedoch nicht alle Werte, die in Abschnitt 2.5 definiert wurden, erreicht werden. Die Gründe liegen bei der zusätzlichen Last, die in Form eines Aluminiumrades an den Motor angeflanscht ist und an der geringen Encoderauflösung sowie an der Zyklusdauer des Reglers.

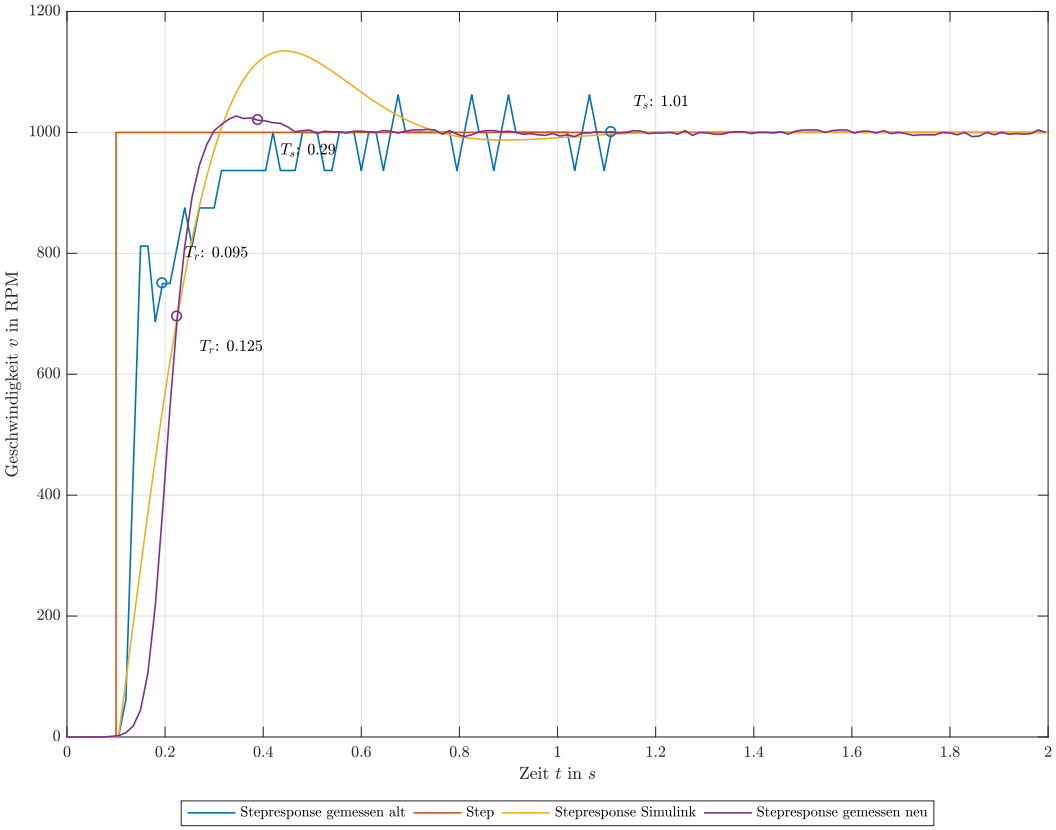

Abbildung 5.3: Closed Loop Stepresponse mit eigenen Reglerparametern
Quelle: Eigene Ausarbeitung

In Abbildung 5.4 ist die Rampe, die im Local-Mode und Chain-Mode gefahren wird, abgebildet. Die „Rampe soll“ Kurve gibt das Profil vor, dem der Regler folgen soll. Die „Rampe ist alt“ stellt die gemessene Drehzahl mit den vom Auftraggeber zur Verfügung gestellten Parametern dar. Im Vergleich dazu ist die „Rampe ist neu“ mit den neuen Regelparametern um das Doppelte schneller. Der Grund für die Abweichung beim Beschleunigen, wird auf das Fehlen des Massenträgheitsmomentes im Modell zurück geführt. Das Minimum und das Maximum liegt bei beiden bei ca. 1818 U/min und 1787 U/min.

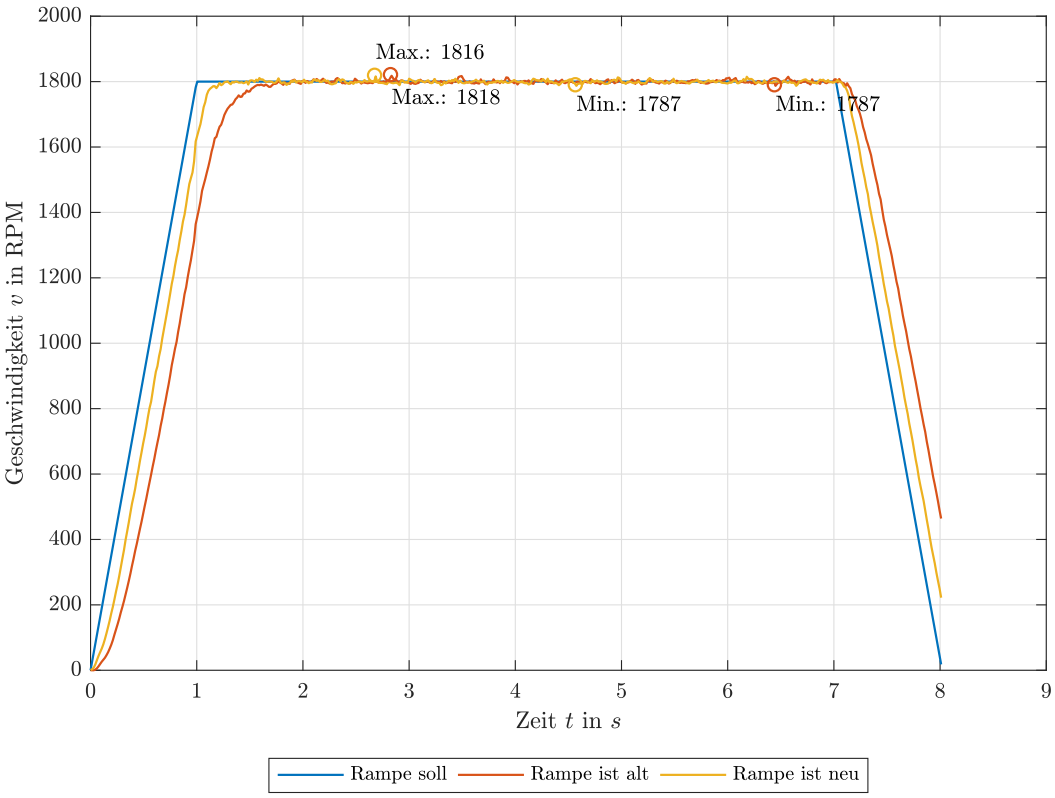

Abbildung 5.4: Rampe Soll Ist Verlgeich
Quelle: Eigene Ausarbeitung

Um die Differenz zwischen „Rampe soll“ und „Rampe ist neu“ zu minimieren, wurden die Beschleunigungs- und Verzögerungswerte so angepasst, dass ein Verzögern bis auf Stillstand möglich ist. Die vom Auftraggeber vorgegebenen Werte werden hierbei nicht verletzt. Das Ergebnis ist in Abbildung 5.5 dargestellt.

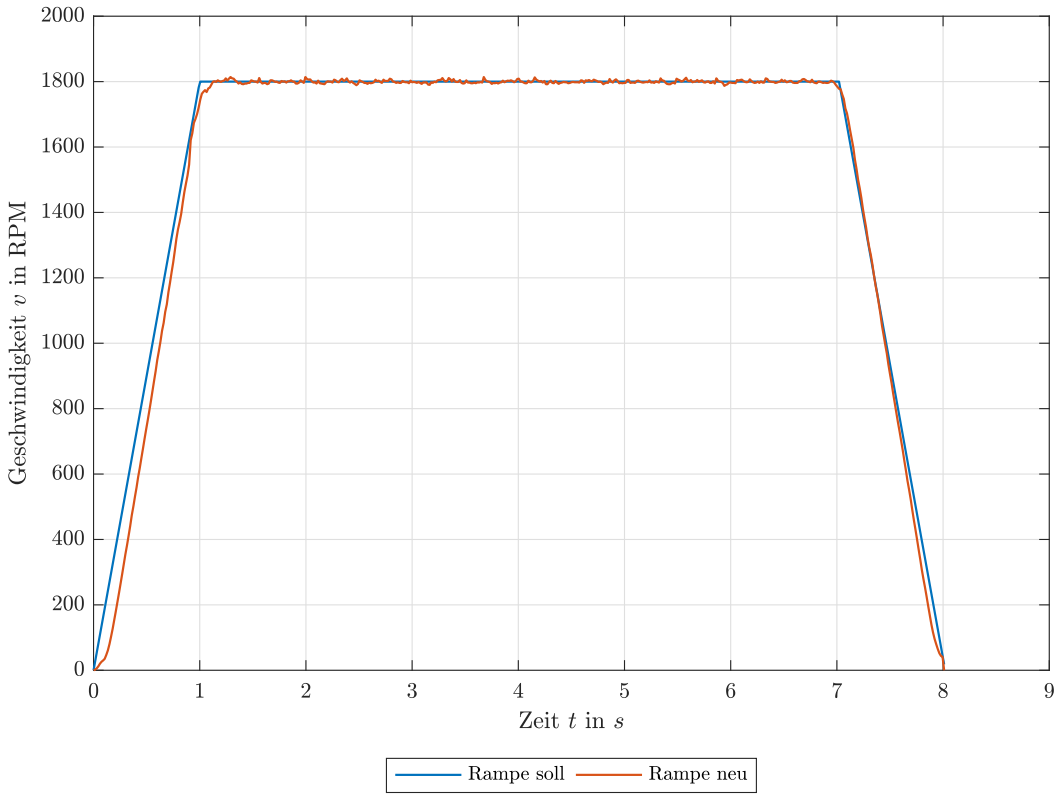

Abbildung 5.5: Rampe Soll Ist Verlgeich mit schnellerer Beschleunigung und Verzögerung
Quelle: Eigene Ausarbeitung

Abkürzungsverzeichnis

CB Conveyor Belt. 4

FR funktionale Anforderung. I, 4, 8–10

IP Internet Protocol. 4

NFR nicht funktionale Anforderung. I, 4, 8–10

PTP Precision Time Protocol. 8

TCP Transmission Control Protocol. 4, 10

Telnet Teletype Network. 1, 4, 6, 8

Literaturverzeichnis

Berend Brouër (1998). Vieweg+Teubner Verlag (siehe S. 11).

Pilsan, Horatiu (2014): *Hardware Access Functions*. Rev 1.6. FH Vorarlberg. Dornbirn, Österreich (siehe S. 8).

Pilsan, Horatiu (2017): *MEM2 Embedded Systems: Project Description*. Rev 1.1. FH Vorarlberg. Dornbirn, Österreich (siehe S. 2).

Anhang

A Display

Abbildung A.1: Display
Quelle: eigene Ausarbeitung

B Tastenfeldbelegung

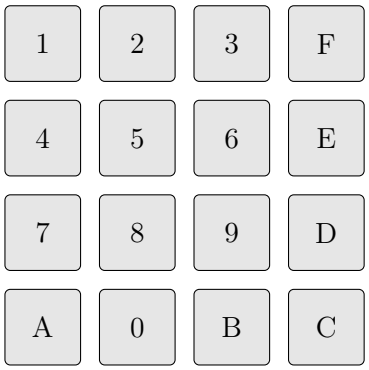

Abbildung B.1: Tastenfeld
Quelle: eigene Ausarbeitung

Keyboard-Taste	**Funktion**
0	nicht belegt
1	nicht belegt
2	erhöhe max. Geschwindigkeit um 100 U/min
3	nicht belegt
4	Start Profil links
5	Start Profil rechts
6	nicht belegt
7	nicht belegt
8	verringere max. Geschwindigkeit um 100 U/min
9	nicht belegt
A	nicht beleg
B	nicht belegt
C	Stopp
D	nicht belegt
E	Auswahl Chain-Mode
F	Auswahl Local-Mode

Tabelle B.1: Funktionen des Tastenfeldes
Quelle: eigene Ausarbeitung

C Kommunikation zwischen den Förderbändern und Master

Befehl	Device	Funktion
„Request\r\n“	linkes Förderband, Master	Anfrage für die Übergabe des Paketes.
„Ready\r\n“	rechtes Förderband, eigenes	Antwort auf die „REQUEST“ Anfrage, wenn kein Paket vorhanden ist.
„Release\r\n“	rechtes Förderband, eigenes	Rückmeldung, dass das Paket erfolgreich übernommen wurde.
„Wait\r\n“	rechtes Förderband, eigenes	Antwort auf die „REQUEST“ Anfrage, wenn das Paket vorhanden ist.
„Right 91.0.0.k\r\n“[a]	Master	IP-Adresse des rechten Förderbandes

Tabelle C.1: Kommunikation zwischen den Förderbändern und Master
Quelle: eigene Ausarbeitung

[a] k steht für eine ganze Zahl von 1-90 und 92-254.

D Telnet Kommunikation

Befehl	**Funktion**
stop	Abbruch des Geschwindigkeitsprofils → Stoppen des Motors
local	Auswahl Local Mode
chain	Auswahl Chain Mode
left	Start Profil links
right	Start Profil rechts
plus	erhöhe max. Geschwindigkeit um 100 U/min
minus	verringere max. Geschwindigkeit um 100 U/min

Tabelle D.1: Telnet Kommunikation
Quelle: eigene Ausarbeitung

E Kalman Filter

Nach folgenden Gleichungen ist der Kalman Filter implementiert[1].

$$p = p + q \tag{E.1}$$

$$k = \frac{p}{p + r} \tag{E.2}$$

$$x = k \cdot (\text{Messwert} - x) \tag{E.3}$$

$$p = (1 - k) \cdot p \tag{E.4}$$

$$p \ldots \text{erwarteter Fehler} \tag{E.5}$$

$$r \ldots \text{Sensorrauschen} \tag{E.6}$$

$$q \ldots \text{Prozessrauschen} \tag{E.7}$$

$$k \ldots \text{Kalman Verstärkung} \tag{E.8}$$

$$x \ldots \text{gefilterter Wert} \tag{E.9}$$

[1] Hier kann der Code heruntergeladen werden: `https://github.com/bachagas/Kalman`.
In `http://interactive-matter.eu/blog/2009/12/18/filtering-sensor-data-with-a-kalman-filter/` ist der 1D Kalman Filter beschrieben.

F Code der Statemachine

```
#include "systemManager.h"

SystemManager * mySystemManager;
StateMachine * myStateMachine;
SystemManager::SystemManager() {
    int countLocalM = 0, LocalM = 0;
    int countChainM = 0, ChainM = 1;
    int countSwitchM = 0, SwitchM = 2;
    int countBeltInMotionM = 0, BetlInMotionM = 3;
    int countPerformingM = 0, PerformingM = 4;
    for (int k = 0; k < sizeof(myBufferSfd) / sizeof(myBufferSfd[0]); ++k) {
        myBufferSfd[k].sfd = -2;
        myBufferSfd[k].wait = false;
        strcpy(myBufferSfd[k].ipAddr, "");
    }

    ctrHandingOver = 0;
    localMode = false;
    chainMode = false;
    orderInProgress = false;
    requestInProcess = false;
    semOrder = semMCreate(SEM_Q_PRIORITY);
    //-----------------------Local-MODE-----------------------------
    tab[LocalM][countLocalM] = new TableEntry("SelectLocalMode", "Idle", "LocalModeS", 0, myAction00, myConditionTrue);
    tab[LocalM][++countLocalM] = new TableEntry("Idle", "Idle", "SetSpeedLocalP", 0, myAction01, myConditionLocalM);
    tab[LocalM][++countLocalM] = new TableEntry("Idle", "Idle", "SetSpeedLocalM", 0, myAction02, myConditionLocalM);
    tab[LocalM][++countLocalM] = new TableEntry("Idle", "BeltInMotion", "BeltInMotionL", 0, myAction03, myConditionLocalM);
    tab[LocalM][++countLocalM] = new TableEntry("Idle", "BeltInMotion", "BeltInMotionR", 0, myAction04, myConditionLocalM);
    tab[LocalM][++countLocalM] = new TableEntry("BeltInMotion", "Idle", "StopBelt", 0, myAction05, myConditionLocalM);
    tab[LocalM][++countLocalM] = new TableEntry("Idle", "SelectLocalMode", "ChainMode", 0, myAction06, myConditionLocalM);
    //-----------------------Chain-MODE-----------------------------
    tab[ChainM][countChainM] = new TableEntry("SelectChainMode", "Idle", "ChainModeS", 0, myAction10, myConditionTrue);
    tab[ChainM][++countChainM] = new TableEntry("Idle", "SelectChainMode", "LocalMode", 0, myAction11, myConditionChainM);
    tab[ChainM][++countChainM] = new TableEntry("Idle", "HandingOverL", "request", 0, myAction12, myConditionChainM);
    tab[ChainM][++countChainM] = new TableEntry("HandingOverL", "BeltInMotion", "constf", 0, myAction13, myConditionChainM);
```

```
36      tab[ChainM][++countChainM] = new TableEntry("BeltInMotion", "
            Waiting", "StopBelt", 0, myAction14, myConditionChainM);
37      tab[ChainM][++countChainM] = new TableEntry("Waiting", "Waiting", "
            wait", 0, myAction15, myConditionChainM);
38      tab[ChainM][++countChainM] = new TableEntry("Waiting", "Idle", "
            CancelOrder", 0, myAction19, myConditionChainM);
39      tab[ChainM][++countChainM] = new TableEntry("Waiting", "
            HandingOverR", "ready", 0, myAction16, myConditionChainM);
40      tab[ChainM][++countChainM] = new TableEntry("HandingOverR", "
            HandingOverR", "constf", 0, myAction17, myConditionChainM);
41      tab[ChainM][++countChainM] = new TableEntry("HandingOverR", "Idle",
             "release", 0, myAction18, myConditionChainM);
42      //-------------------Switch-MODE-------------------
43      tab[SwitchM][countSwitchM] = new TableEntry("startSystem", "Init",
            "InitSystem", 0, myAction20, myConditionTrue);
44      tab[SwitchM][++countSwitchM] = new TableEntry("Init", "Init", "
            Error", 0, myAction21, myConditionTrue);
45      tab[SwitchM][++countSwitchM] = new TableEntry("Init", "ChainMode",
            "ChainMode", 0, myAction22, myConditionTrue);
46      tab[SwitchM][++countSwitchM] = new TableEntry("Init", "LocalMode",
            "LocalMode", 0, myAction23, myConditionTrue);
47      tab[SwitchM][++countSwitchM] = new TableEntry("LocalMode", "
            ChainMode", "ChainModeL", 0, myAction24, myConditionTrue);
48      tab[SwitchM][++countSwitchM] = new TableEntry("ChainMode", "
            LocalMode", "LocalModeC", 0, myAction25, myConditionTrue);
49      //-------------------Belt-In-Motion-------------------
50      tab[BetlInMotionM][countBeltInMotionM] = new TableEntry("startBelt"
            , "RampUp", "BeltInMotionL", 0, myAction30, myConditionTrue);
51      tab[BetlInMotionM][++countBeltInMotionM] = new TableEntry("
            startBelt", "RampUp", "BeltInMotionR", 0, myAction30,
            myConditionTrue);
52      tab[BetlInMotionM][++countBeltInMotionM] = new TableEntry("RampUp",
             "startBelt", "StopBelt", 0, myAction31, myConditionTrue);
53      tab[BetlInMotionM][++countBeltInMotionM] = new TableEntry("RampUp",
             "ConstantVelocity", "rampupf", 0, myAction32, myConditionTrue);
54      tab[BetlInMotionM][++countBeltInMotionM] = new TableEntry("
            ConstantVelocity", "startBelt", "StopBelt", 0, myAction33,
            myConditionTrue);
55      tab[BetlInMotionM][++countBeltInMotionM] = new TableEntry("
            ConstantVelocity", "RampDown", "constf", 0, myAction34,
            myConditionTrue);
56      tab[BetlInMotionM][++countBeltInMotionM] = new TableEntry("RampDown
            ", "startBelt", "StopBelt", 0, myAction35, myConditionTrue);
57      tab[BetlInMotionM][++countBeltInMotionM] = new TableEntry("RampDown
            ", "startBelt", "rampdownf", 0, myAction36, myConditionTrue);
58      //-------------------Performing-------------------
59      tab[PerformingM][countPerformingM] = new TableEntry("
            startPerforming", "Performing", "request", 0, myAction40,
            myConditionChainM);
60      tab[PerformingM][++countPerformingM] = new TableEntry("Performing",
             "Performing", "request", 0, myAction41, myConditionChainM);
61      tab[PerformingM][++countPerformingM] = new TableEntry("Performing",
             "startPerforming", "release", 0, myAction42, myConditionChainM)
            ;
62
63      // Initialize timer names for all diagrams
64      // Timer names are always Timer followed by the diagram number
65      timerNames[ChainM] = "Timer1";
66      timerNames[BetlInMotionM] = "Timer3";
67
```

```
68      // Initialize line numbers for all diagrams
69      lines[LocalM] = countLocalM + 1;
70      lines[ChainM] = countChainM + 1;
71      lines[SwitchM] = countSwitchM + 1;
72      lines[BetlInMotionM] = countBeltInMotionM + 1;
73      lines[PerformingM] = countPerformingM + 1;
74
75      // Initialize first state for all diagrams
76      actualState[LocalM] = "SelectLocalMode";
77      actualState[ChainM] = "SelectChainMode";
78      actualState[SwitchM] = "startSystem";
79      actualState[BetlInMotionM] = "startBelt";
80      actualState[PerformingM] = "startPerforming";
81
82      // Set the actual number of diagrams
83      diagrams = PerformingM + 1;
84
85      // Create instance of state machine
86      myStateMachine = new StateMachine;
87
88      // Start timer for each diagram which needs one in the first state!
89      myStateMachine->diaTimerTable[ChainM]->startTimer(tab[ChainM][0]->
            eventTime);
90      myStateMachine->diaTimerTable[BetlInMotionM]->startTimer(tab[
            BetlInMotionM][0]->eventTime);
91      return;
92  }
93
94  SystemManager::~SystemManager() {
95      return;
96  }
97
98  void SystemManager::requestEvent(Cmd command) {
99      printf("requestEvent im SystemManager %i\n\r", command);
100     switch (command) {
101     case CHAIN:
102         myStateMachine->sendEvent("ChainMode");
103         break;
104     case LOCALM:
105         myStateMachine->sendEvent("LocalMode");
106         break;
107     case PLUS:
108         myStateMachine->sendEvent("SetSpeedLocalP");
109         break;
110     case MINUS:
111         myStateMachine->sendEvent("SetSpeedLocalM");
112         break;
113     case LEFT:
114         if (localMode)
115             myStateMachine->sendEvent("BeltInMotionL");
116         break;
117     case RIGHT:
118         if (localMode)
119             myStateMachine->sendEvent("BeltInMotionR");
120         break;
121     case STOP:
122         if (localMode) {
123             myStateMachine->sendEvent("StopBelt");
124         } else {
125             myStateMachine->sendEvent("CancelOrder");
```

```
126                 printf(" Stopping in Chain-Mode not allowed!");
127             }
128             break;
129         case REQUEST:
130             myStateMachine->sendEvent("request");
131             break;
132         case READY:
133             myStateMachine->sendEvent("ready");
134             break;
135         case RELEASE:
136             myStateMachine->sendEvent("release");
137             break;
138         case WAIT:
139             myStateMachine->sendEvent("wait");
140             break;
141         case DELETE:
142 //          deinitSystem();
143             break;
144         case FINISHED:
145             myStateMachine->sendEvent("StopBelt");
146             break;
147         case NOTFOUND:
148             myStateMachine->sendEvent("Error");
149             break;
150         default:
151             myStateMachine->sendEvent("Error");
152             break;
153     }
154 }
155
156 void SystemManager::requestEvent(Cmd command, int sFd) {
157     /*  */
158     int j = 0;
159     int k = 0;
160     int ctr = 0;
161     bool breakIt = false;
162     int strCmp = 0;
163     if (command == REQUEST && !localMode) {
164         printf("sFd im sysM: %i requestInProcess %i\n", sFd,
                requestInProcess);
165         struct sockaddr_in addr;
166         socklen_t addr_size = sizeof(struct sockaddr_in);
167         int res = getpeername(sFd, (struct sockaddr *) &addr, &
                addr_size);
168         char *clientip = new char[20];
169         strcpy(clientip, inet_ntoa(addr.sin_addr));
170         printf("clientip %s\n", clientip);
171         semTake(semOrder, WAIT_FOREVER);
172         for (j = 0; j < sizeof(myBufferSfd) / sizeof(myBufferSfd[0]); +
                +j) {
173             /* selber auftrag mit zaehlen */
174             if(myBufferSfd[j].sfd == sFd){
175                 ctr ++;
176             }
177             /* auftrag bereits da, wenn in der Uebergabe ist ? */
178             /* auftrag bereits da, vom selben Client ? -> Abbruch!!! */
179             strCmp = strcmp(myBufferSfd[j].ipAddr, clientip);
180             if (strCmp == 0) {
181                 mySystemManager->myTCPServer->send(NOTFOUND, sFd);
182                 breakIt = true;
```

```
183                 break;
184             }
185             /* auftrag bereits zwei mal da -> nicht annehmen! */
186             if(ctr > 0){
187                 mySystemManager->myTCPServer->send(NOTFOUND, sFd);
188                 breakIt = true;
189                 break;
190             }
191             if (myBufferSfd[j].sfd == -2) {
192                 if (j == 0) {
193                     orderInProgress = true;
194                 }
195                 myBufferSfd[j].sfd = sFd;
196                 myBufferSfd[j].wait = j;
197                 strcpy(myBufferSfd[j].ipAddr, inet_ntoa(addr.sin_addr))
                        ;
198                 break;
199             } else {
200                 k++;
201             }
202         }
203         if (j > k) {
204             perror("Auftragspuffer voll!\n");
205             breakIt = true;
206         }
207         semGive(semOrder);
208     }
209     if (!breakIt)
210         requestEvent(command);
211 }
212
213 int SystemManager::getActualSfd() {
214 //  return actualSfd;
215     return myBufferSfd[0].sfd;
216 }
217 int SystemManager::getWaitingSfd() {
218     int sfd = -2;
219     int j;
220     for (j = 0; j < sizeof(myBufferSfd) / sizeof(myBufferSfd[0]); ++j)
            {
221         if (myBufferSfd[j].wait) {
222             sfd = myBufferSfd[j].sfd;
223             myBufferSfd[j].wait = false; /* wait send ! */
224             break;
225         }
226     }
227     return sfd;
228 }
229
230 void SystemManager::deinitSystem() {
231     printf("deinitSystem called \n");
232     myDisplay->deinitTask();
233     myKeyboard->deinitTask();
234     myPTPClient->stopPTPClient();
235     myTCPClient->stopClient();
236     delete (myKeyboard);
237     beLazy(500);
238     printf("deinitSystem called 6 \n");
239     delete (myDisplay);
240     beLazy(500);
```

```
241     printf("deinitSystem called 7 \n");
242 #ifdef __COM
243     delete myTelnetServer;
244     delete myTCPServer;
245 #endif
246     delete myTCPClient;
247     printf("deinitSystem called 8 \n");
248     delete (myPTPClient);
249 
250     delete (myStateMachine);
251     delete (mySystemManager);
252     printf("deinitSystem called last! \n");
253 }
254 
255 void SystemManager::initSystem() {
256     // Set local IP address according to MAC table
257     setLocalIp();
258     initHardware(0);
259     // Create instance of my Keyboard
260     myKeyboard = new Keyboard(mySystemManager);
261     myKeyboard->initTask();
262 
263     myDisplay = new Display(mySystemManager, myStateMachine);
264     myDisplay->initTask();
265 
266     myVelocityChain = new VelocityProfile(925, 925, 6000, 1800, false);
267     myVelocityHandingOver = new VelocityProfile(1, 1, 1000, 100, false)
        ;
268     myVelocityLocal = new VelocityProfile(925, 925, 6000, 1600, false);
269 //  myVelocityProfile = new VelocityProfile(1, 1, 1000, 1000, false);
270 
271     myTelnetServer = new TcpServer(mySystemManager);
272     myTelnetServer->setReplyMsgSize(REPLY_MSG_SIZE);
273     myTelnetServer->setRequestMsgSize(REQUEST_MSG_SIZE);
274     myTelnetServer->setMaxConnections(TELNET_MAX_CONNECTIONS);
275     myTelnetServer->setPort(TELNET_PORT_NUM);
276     myTelnetServer->setStackSize(TELNET_STACK_SIZE);
277     myTelnetServer->setWorkTaskPrio(TELNET_WORK_PRIORITY);
278     myTelnetServer->setType(TELNET);
279     myTelnetServer->setTaskName("tTcpTelnet");
280     myTelnetServer->initTask();
281 #define __COM
282 #ifdef __COM
283     myTCPServer = new TcpServer(mySystemManager);
284     myTCPServer->setReplyMsgSize(REPLY_MSG_SIZE);
285     myTCPServer->setRequestMsgSize(REQUEST_MSG_SIZE);
286     myTCPServer->setMaxConnections(TCP_MAX_CONNECTIONS);
287     myTCPServer->setPort(TCP_PORT_NUM);
288     myTCPServer->setStackSize(TCP_STACK_SIZE);
289     myTCPServer->setWorkTaskPrio(TCP_WORK_PRIORITY);
290     myTCPServer->setType(SERVER_CLIENT);
291     myTCPServer->setTaskName("tTcpServer");
292     myTCPServer->initTask();
293 #endif
294 
295     myPTPClient = new PTPClient;
296     myPTPClient->setTaskName("tPtpClient");
297     myPTPClient->startPTPClient();
298 
299     myTCPClient = new TcpClient(mySystemManager);
```

```
300     myTCPClient->setReplyMsgSize(REPLY_MSG_SIZE);
301     myTCPClient->setRequestMsgSize(REQUEST_MSG_SIZE);
302     myTCPClient->setMaxConnections(TCP_CLIENT_MAX_CONNECTIONS);
303     myTCPClient->setPort(TCP_CLIENT_PORT_NUM);
304     myTCPClient->setStackSize(TCP_CLIENT_STACK_SIZE);
305     myTCPClient->setWorkTaskPrio(TCP_CLIENT_WORK_PRIORITY);
306     myTCPClient->setTaskName("tTcpClient");
307     myTCPClient->setType(SERVER_CLIENT);
308
309     myController = new Controller(mySystemManager, myStateMachine);
310     myController->controllerRun();
311
312 }
313
314 void SystemManager::connectToServer(char * serverName) {
315     myTCPClient->setServerName(serverName);
316     myTCPClient->tcpClientStart();
317     myDisplay->setInitialization(serverName);
318 }
319
320 void myAction00() {
321     printf(" SelectLocalMode -> Transition00 -> Idle\n\r");
322     mySystemManager->myDisplay->setOperationMode("Local-Mode");
323     mySystemManager->myDisplay->setStatus("Idle");
324     mySystemManager->myDisplay->setMaxSpeed(mySystemManager->
            myVelocityLocal->getA());
325     return;
326 }
327 void myAction01() {
328     printf(" Idle -> Transition01 -> Idle\n\r");
329     mySystemManager->myVelocityLocal->setA(mySystemManager->
            myVelocityLocal->getA() + 100);
330     if (mySystemManager->myVelocityLocal->getA() > 2200) {
331         mySystemManager->myVelocityLocal->setA(2200);
332     }
333     mySystemManager->myDisplay->setMaxSpeed(mySystemManager->
            myVelocityLocal->getA());
334     return;
335 }
336 void myAction02() {
337     printf(" Idle -> Transition02 -> Idle\n\r");
338     mySystemManager->myVelocityLocal->setA(mySystemManager->
            myVelocityLocal->getA() - 100);
339     if (mySystemManager->myVelocityLocal->getA() < 1000) {
340         mySystemManager->myVelocityLocal->setA(1000);
341     }
342     mySystemManager->myDisplay->setMaxSpeed(mySystemManager->
            myVelocityLocal->getA());
343     return;
344 }
345 void myAction03() {
346     printf(" Idle -> Transition03 -> BeltInMotion\n\r");
347     mySystemManager->myVelocityLocal->setDirection(true);
348     mySystemManager->myController->setProfile(mySystemManager->
            myVelocityLocal);
349     mySystemManager->myDisplay->setDirection(mySystemManager->
            myVelocityLocal->getDirection());
350     mySystemManager->myDisplay->setStatus("Belt in Motion");
351     return;
352 }
```

```
353 void myAction04() {
354     printf(" Idle -> Transition04 -> BeltInMotion\n\r");
355     mySystemManager->myVelocityLocal->setDirection(false);
356     mySystemManager->myController->setProfile(mySystemManager->
            myVelocityLocal);
357     mySystemManager->myDisplay->setDirection(mySystemManager->
            myVelocityLocal->getDirection());
358     mySystemManager->myDisplay->setStatus("Belt in Motion");
359     return;
360 }
361 void myAction05() {
362     printf(" BeltInMotion -> Transition05 -> Idle\n\r");
363     mySystemManager->myDisplay->setStatus("Idle");
364     mySystemManager->myController->setSpeedToZero();
365     return;
366 }
367 void myAction06() {
368     myStateMachine->sendEvent("ChainModeL");
369     printf(" Idle -> Transition06 -> SelectLocalMode\n\r");
370     return;
371 }
372
373 void myAction10() {
374     printf(" SelectChainMode -> Transition10 -> Idle\n\r");
375     mySystemManager->myDisplay->setOperationMode("Chain-Mode");
376     mySystemManager->myDisplay->setMaxSpeed(mySystemManager->
            myVelocityChain->getA());
377     mySystemManager->myDisplay->setDirection(mySystemManager->
            myVelocityChain->getDirection());
378     mySystemManager->myDisplay->setStatus("Idle");
379     return;
380 }
381
382 void myAction11() {
383     printf(" Idle -> Transition11 -> SelectChainMode\n\r");
384     myStateMachine->sendEvent("LocalModeC");
385     return;
386 }
387 void myAction12() {
388     printf(" Idle -> Transition12 -> HandingOverL\n\r");
389     mySystemManager->requestInProcess = true;
390     mySystemManager->orderInProgress = true;
391     printf("myAcion12() %i\n", mySystemManager->getActualSfd());
392     mySystemManager->myTCPServer->send(READY, mySystemManager->
            getActualSfd());
393     mySystemManager->myDisplay->setStatus("Handing over LeftN");
394     mySystemManager->myController->setProfile(mySystemManager->
            myVelocityHandingOver);
395     mySystemManager->myController->setConstantVelocity();
396     return;
397 }
398 void myAction13() {
399     printf(" HandingOverL -> Transition13 -> BeltInMotion\n\r");
400     struct timespec timeStamp;
401     mySystemManager->myController->setProfile(mySystemManager->
            myVelocityChain);
402     myStateMachine->sendEvent("BeltInMotionR");
403     printf("myAcion13() %i\n", mySystemManager->getActualSfd());
404     mySystemManager->orderInProgress = false;
```

```
405     mySystemManager->myTCPServer->send(RELEASE, mySystemManager->
            getActualSfd());
406     mySystemManager->myDisplay->setStatus("Belt in Motion");
407     clock_gettime(CLOCK_REALTIME, &timeStamp);
408     mySystemManager->myDisplay->setRecTimestamp(&timeStamp);
409     mySystemManager->myController->setSpeedToZero();
410     return;
411 }
412 void myAction14() {
413     printf(" BeltInMotion -> Transition14 -> Waiting\n\r");
414     mySystemManager->myTCPClient->send(REQUEST);
415     mySystemManager->myController->setSpeedToZero();
416     mySystemManager->myDisplay->setStatus("Waiting");
417     return;
418 }
419 void myAction15() {
420     printf(" Waiting -> Transition15 -> Waiting\n\r");
421     return;
422 }
423 void myAction16() {
424     printf(" Waiting -> Transition16 -> HandingOverR\n\r");
425     mySystemManager->myDisplay->setStatus("Handing over RightN");
426     mySystemManager->myController->setProfile(mySystemManager->
            myVelocityHandingOver);
427     mySystemManager->myController->setConstantVelocity();
428     return;
429 }
430 void myAction17() {
431     printf(" HandingOverR -> Transition17 -> HandingOverR\n\r");
432     char temp[30];
433
434     sprintf(temp, "Handing over RightN: %i", mySystemManager->
            ctrHandingOver);
435     mySystemManager->myDisplay->setStatus(temp);
436     mySystemManager->myController->setProfile(mySystemManager->
            myVelocityHandingOver);
437     mySystemManager->myController->setConstantVelocity();
438     mySystemManager->ctrHandingOver++;
439     return;
440 }
441 void myAction18() {
442     printf(" HandingOverR -> Transition18 -> Idle\n\r");
443     struct timespec timeStamp;
444     mySystemManager->myDisplay->setStatus("Idle");
445     clock_gettime(CLOCK_REALTIME, &timeStamp);
446     mySystemManager->myDisplay->setSendTimestamp(&timeStamp);
447     mySystemManager->myController->setSpeedToZero();
448     mySystemManager->requestInProcess = false;
449     mySystemManager->ctrHandingOver = 0;
450     return;
451 }
452
453 void myAction19() {
454     printf(" Waiting -> Transition19 -> Idle\n\r");
455     myStateMachine->sendEvent("release");
456     mySystemManager->requestInProcess = false;
457     mySystemManager->myDisplay->setStatus("Idle");
458
459 }
460
```

```
461 void myAction20() {
462     printf(" StartSystem -> Transition20 -> Init\n\r");
463     mySystemManager->initSystem();
464     mySystemManager->myDisplay->setOperationMode("SelectMode");
465     return;
466 }
467 void myAction21() {
468     printf(" Init -> Transition21 -> Error\n\r");
469     return;
470 }
471 void myAction22() {
472     printf(" Init -> Transition22 -> ChainMode\n\r");
473     mySystemManager->localMode = false;
474     mySystemManager->chainMode = true;
475     myStateMachine->sendEvent("ChainModeS");
476     return;
477 }
478 void myAction23() {
479     printf(" Init -> Transition23 -> LocalMode\n\r");
480     mySystemManager->localMode = true;
481     mySystemManager->chainMode = false;
482     myStateMachine->sendEvent("LocalModeS");
483     return;
484 }
485 void myAction24() {
486     printf(" LocalMode -> Transition24 -> ChainMode\n\r");
487     mySystemManager->localMode = false;
488     mySystemManager->chainMode = true;
489     myStateMachine->sendEvent("ChainModeS");
490     return;
491 }
492 void myAction25() {
493     printf(" ChainMode -> Transition25 -> LocalMode\n\r");
494     mySystemManager->localMode = true;
495     mySystemManager->chainMode = false;
496     myStateMachine->sendEvent("LocalModeS");
497     return;
498 }
499
500 void myAction30() {
501     printf(" StartBelt -> Transition30 -> RampUp\n\r");
502     mySystemManager->myController->Subsystem_initialize();
503     mySystemManager->myController->setRampUp();
504     return;
505 }
506 void myAction31() {
507     printf(" RampUp -> Transition31 -> startBelt\n\r");
508     mySystemManager->myController->setSpeedToZero();
509     return;
510 }
511 void myAction32() {
512     printf(" RampUp -> Transition32 -> ConstantVelocity\n\r");
513     mySystemManager->myController->setConstantVelocity();
514     return;
515 }
516 void myAction33() {
517     printf(" ConstantVelocity -> Transition33 -> startBelt\n\r");
518     mySystemManager->myController->setSpeedToZero();
519     return;
520 }
```

```
521 void myAction34() {
522     printf(" ConstantVelocity -> Transition34 -> RampDown\n\r");
523     mySystemManager->myController->setRampDown();
524     return;
525 }
526 void myAction35() {
527     printf(" RampDown -> Transition35 -> startBelt\n\r");
528     mySystemManager->myController->setSpeedToZero();
529     return;
530 }
531 void myAction36() {
532     printf(" RampDown -> Transition36 -> startBelt\n\r");
533     Cmd command = FINISHED;
534     mySystemManager->myController->setSpeedToZero();
535     mySystemManager->requestEvent(command);
536     return;
537 }
538
539 void myAction40() {
540     printf(" startPerforming -> Transition40 -> Performing\n\r");
541     return;
542 }
543 void myAction41() {
544     printf(" Performing -> Transition41 -> Performing\n\r");
545     int j;
546     mySystemManager->myTCPServer->send(WAIT, mySystemManager->
            getWaitingSfd());
547     return;
548 }
549 void myAction42() {
550     printf(" Performing -> Transition42 -> startPerforming\n\r");
551     int j;
552     semTake(semOrder, WAIT_FOREVER);
553     for (j = 0; j < sizeof(mySystemManager->myBufferSfd) / sizeof(
            mySystemManager->myBufferSfd[0]); ++j) {
554         swap(mySystemManager->myBufferSfd[j], mySystemManager->
                myBufferSfd[j + 1]);
555     }
556     mySystemManager->myBufferSfd[j - 1].sfd = -2;
557     mySystemManager->myBufferSfd[j - 1].wait = false;
558     semGive(semOrder);
559     strcpy(mySystemManager->myBufferSfd[j].ipAddr, "");
560     if (mySystemManager->getActualSfd() > 0) {
561         mySystemManager->requestEvent(REQUEST);
562     }
563 }
564
565 bool myConditionTrue() {
566     return TRUE;
567 }
568
569 bool myConditionLocalM() {
570     return mySystemManager->localMode;
571 }
572
573 bool myConditionChainM() {
574     return mySystemManager->chainMode;
575 }
```

Listing F.1: Statemachine mit den Membermethoden