FH Vorarlberg
University of Applied Sciences

Embedded System 2

Dokumentation

Fachhochschule Vorarlberg
MEM2

Betreut von:
Horatiu O. Pilsan

Vorgelegt von:
Emine Kaya
Walter R. Schierl

Dornbirn, 29.07.2017

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Anforderungen

1.1 Textuelle Spezifikation der Anforderungen des Systems

1.1.1 Local Service Operation Mode

Folgende Anforderungen sind für die Funktionalität im „Local Service Operation Mode“ notwendig und daher mit FRL (Functional Requirement LOCAL-Mode) gekennzeichnet. Anforderungen welche nicht für die Funktion notwendig sind, werden mit NFRL (Non Functional Requirement LOCAL-Mode) bezeichnet.

Tabelle 1: Anforderungen für Local Service Operation Mode

Local Service Operation Mode			
FRL1	Förderband	muss	manuell durch Bediener steuerbar sein (keine Netzwerkkommunikation)
FRL2	Förderband	muss	dem definierten Bewegungsprofil folgen, das heißt bis zur gewünschten Drehzahl benötigt dieser 1s für die Steigung und 1s für den Stillstand.
FRL3	Richtung	muss	nach links oder rechts frei wählbar sein
NFRL1	Ziel-Drehzahl	muss	zwischen 1000-2200rpm in 100rpm Schritten liegen

Das Förderband kann im “Local Service Operation Mode” Pakete in beide Richtungen befördern. Dabei kann die Motordrehzahl des Förderbandes im Bereich von [1000…2200] rpm in Schritten von 100 rpm verändert werden.

1.1.2 Chain Operation Mode

Folgende Anforderungen sind für die Funktionalität im „Chain Service Operation Mode“ notwendig und daher mit FRC (Functional Requirement CHAIN-Mode) gekennzeichnet. Anforderungen welche nicht für die Funktion notwendig sind, werden mit NFRC (Non Functional Requirement CHAIN-Mode) bezeichnet.

Im “Chain Operation Mode“ werden die Pakete nur von links nach rechts transportiert. Wichtig ist hierbei, dass die Motordrehzahl per Default 1800 rpm beträgt und das Befördern eines Paketes von links nach rechts eine Zeit von 8 Sekunden in Anspruch nimmt.

Für diesen Ablauf frägt das linke Förderband mit dem REQUEST-Zeichen ab, ob das folgende Förderband frei ist oder nicht. Wenn das folgende Förderband sich im IDLE-Zustand befindet, wird ein READY-Zeichen vom angefragten Förderband zurückgesendet. Für den Fall, dass sich schon ein Paket auf dem angefragten Förderband befindet, wird ein WAIT-Zeichen zurückgegeben.

Wenn das angefragte Förderband belegt war und nun frei wird, sendet es ein READY-Signal an das vorherige Förderband, welches sich zu seiner linken Seite befindet. Nun starten beide Förderbänder das Payloadpassing für die Übergabe des Pakets.

Tabelle 2: Anforderungen von Chain Operation Mode

Chain Operation Mode			
FRC1	Förderband	muss	in einer geschlossenen Schleife das Packet übergeben können (Netzwerkkommunikation erforderlich)
FRC2	Förderband	muss	mit dem definierten Bewegungsprofil das Paket zum nächsten Förderband befördern können
NFRC1	Bewegungsprofil	muss	der Abbildung und der Beschreibung aus dem Abschnitt „CHAIN-Mode“ entsprechen
NFRC2	Förderband	sollte	nur in eine Richtung Pakete befördern können (somit vorwärts)
NFRC3	Kommunikation	muss	über TCP/IP und definierten Protokoll(XML,YAML, Jason,..) erfolgen
NFRC4	Paketübernahme	muss	mit den vorgegebenen Nachrichtentypen erfolgen (REQUEST, WAIT, READY, RELEASE)
NFRC5	Jedes Förderband	muss	bei der Übernahme eines Paketes leer sein
NFRC6	Förderband	muss	mit der REQUEST Nachricht beim nächsten Förderband angekündigt werden, ob es frei ist oder nicht
NFRC7	Förderband	muss	eine WAIT Nachricht nach links senden, wenn das Band auf REQUEST Anfrage belegt ist
NFRC8	Förderband	muss	Nachdem das Paket erhalten wurde, wird eine READY Nachricht an das linke Förderband gesendet
NFRC9	Paketübergabe	sollte	nach einer READY Nachricht vom rechten Förderband starten und nach Empfang einer RELEASE Nachricht die Übergabe des Paketes beenden

Es befindet sich keine Lichtschranke am Ende jedes Förderbandes, welche erfassen könnte ob das Paket angekommen ist bzw. an das nächste Förderband übergeben wurde. Daher bewegt sich das Paket mit einer Paketübergabezeit von 1 Sekunde mit 100rpm nach rechts, um die Übergabe zwischen zwei Förderbänder zu realisieren.

Daraufhin informiert das rechte Förderband das linke Förderband mit dem RELEASE-Zeichen, dass das Paket erfolgreich übergeben und erhalten wurde. Das linke Förderband stoppt daraufhin und kehrt in den IDLE-Zustand zurück. Somit kann das Paket, wenn das nächste Förderband frei ist, mit dem gleichen Ablauf bzw. REQUEST-Anfrage weiterbefördert werden, oder bleibt stehen, da das nächste Förderband noch belegt ist.

1.1.3 Zusätzliche Anforderungen

Folgende Anforderungen sind zusätzlich Anforderungen und sind für die Funktion nicht notwendig und werden daher mit NFRA (Non Functional Requirement Additional) bezeichnet.

Tabelle 3: Zusätzliche Anforderungen

Zusätzliche Anforderungen			
NFRA1	Drehzahl	muss	in einer geschlossenen Regelschleife geregelt werden (PI-Regler)
NFRA2	Drehzahl	muss	die Toleranz von ± 36 rpm einhalten
NFRA3	Tastenfeld	muss	dem Bediener als Eingabemöglichkeit zu Verfügung stehen
NFRA4	Telnetverbindung	muss	dem Bediener als Eingabemöglichkeit zu Verfügung stehen
NFRA5	HwFunc.pdf	sollte	zur Beschreibung der notwendigen Funktionen zur Verwendung des Mikrocontroller-Boards verwendet werden (ILIAS)
NFRA6	Display	muss	wichtige Informationen über den Zustand des Systems anzeigen
NFRA9	Am Display	sollte	ein Zeitstempel erscheinen, welcher die Ankunftszeit des Paketes wiedergibt.

Die Drehzahl des Motors sollte mittels Drehzahlrückführung und PI-Regler geregelt werden. Durch die Regelgüte kann eine Aussage über die Qualität der Regelung gemacht werden. Eine Regelabweichung von ca. ± 36 rpm (2%) ist dabei erlaubt. Die Obergrenze für das Überschwingen (c_{max}) sollte 5% nicht überschreiten.

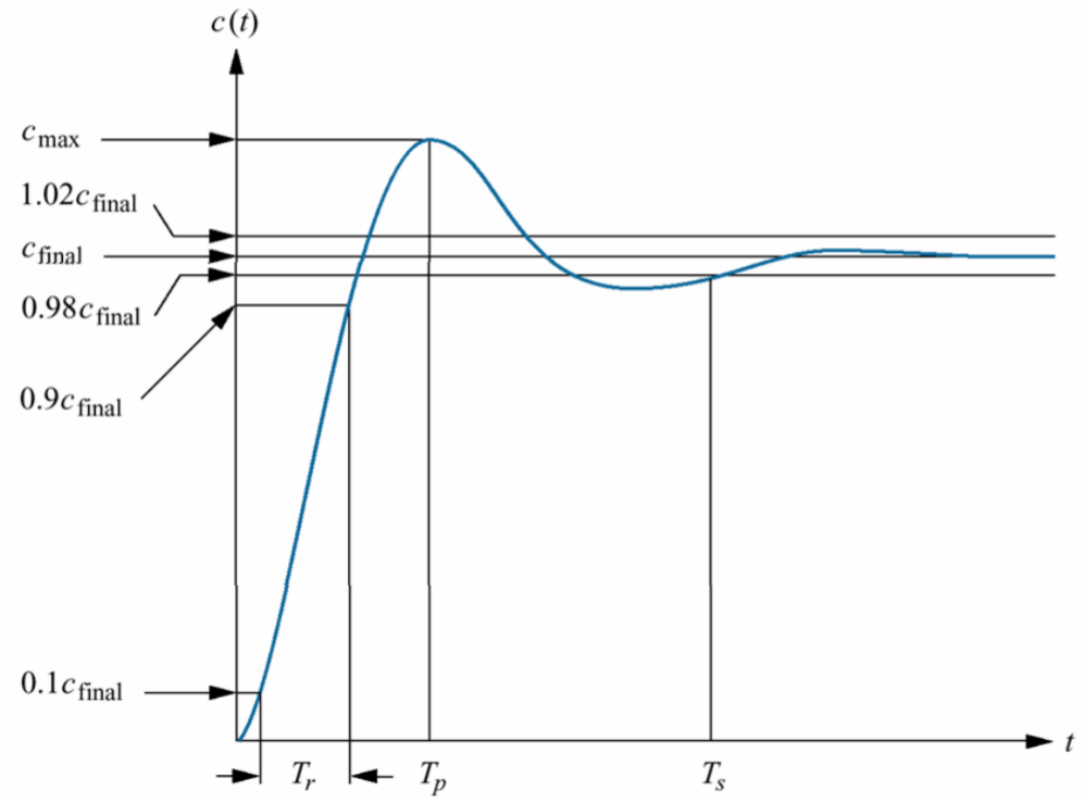

Abbildung 1: Sprungantwort

Folgende Systemzeiten wie die Risetime ($T_r = 5ms$), Peaktime ($T_p = 7ms$) und die Settletime ($T_s = 15ms$) dürfen in der Sprungantwort nicht überschritten werden.

Es sollte möglich sein, das Förderband entweder über die lokale Tastatur oder über eine Telnet-Verbindung zu bedienen. Wichtige Informationen über den Zustand des Systems sollten auf dem Display angezeigt werden. Es ist zu berücksichtigen, dass Anforderungsänderungen auftreten können.

1.2 Use-Case-Diagramm

Im Laufe der Projektphase und Programmierung hat sich ein tieferes Verständnis bezogen auf die Use-Cases für die beiden Modi „Local-Mode“ und „Chain-Mode“ ergeben was auch im Update der Use-Cases Abbildung 3 und Abbildung 5 ersichtlich ist.

Lokal Service Operation Mode:

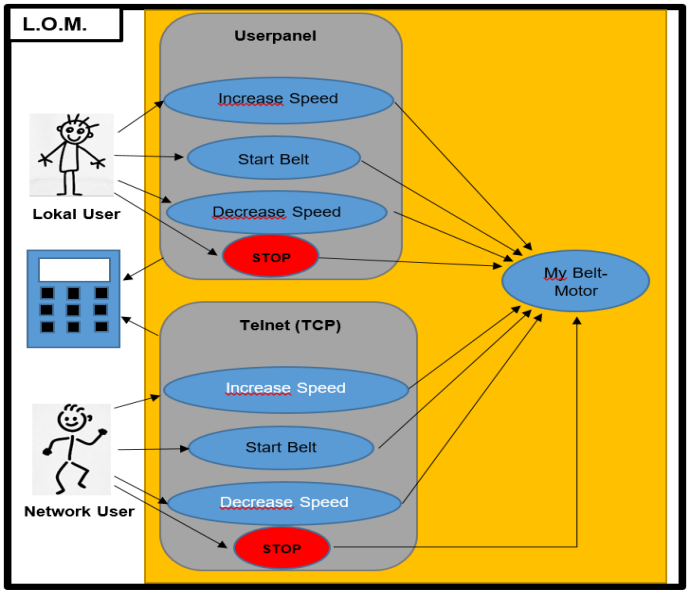

Abbildung 2: Use-Case Diagram Local Service Operation Mode

Update: Lokal Service Operation Mode

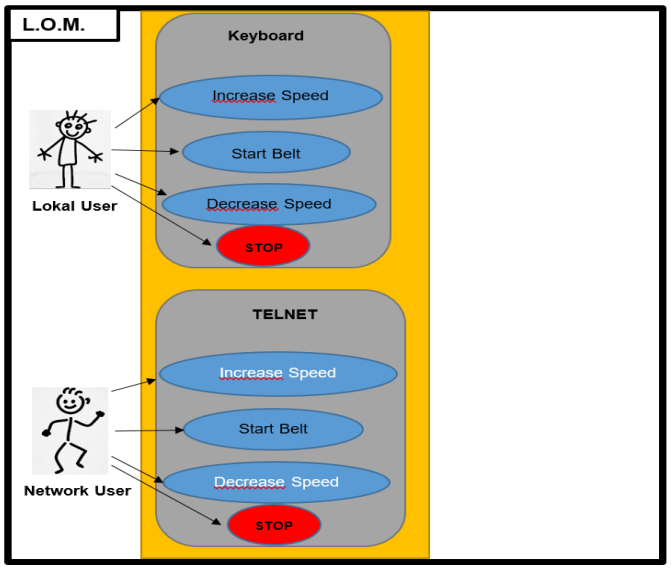

Abbildung 3: UPDATE - Use-Case Diagram Local Service Operation Mode

Chain Operation Mode:

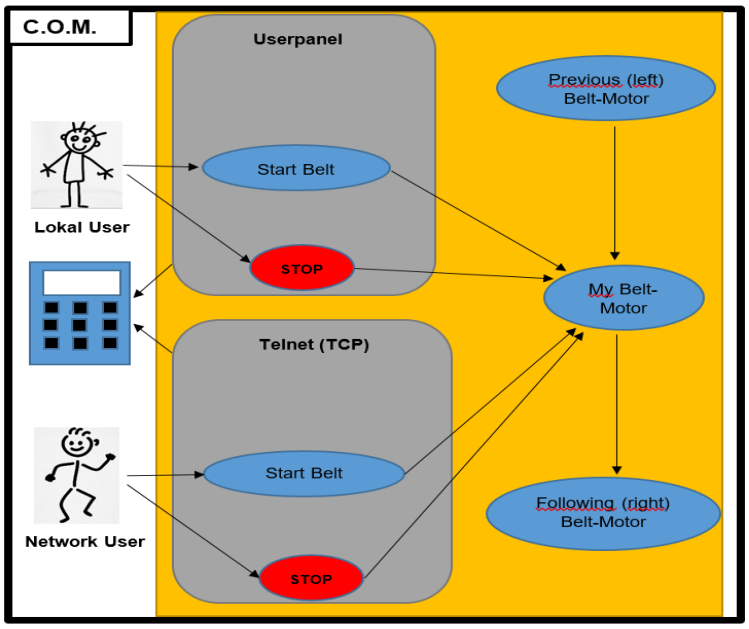

Abbildung 4: Use-Case Diagram Chain Operation Mode

Update: Chain Operation Mode

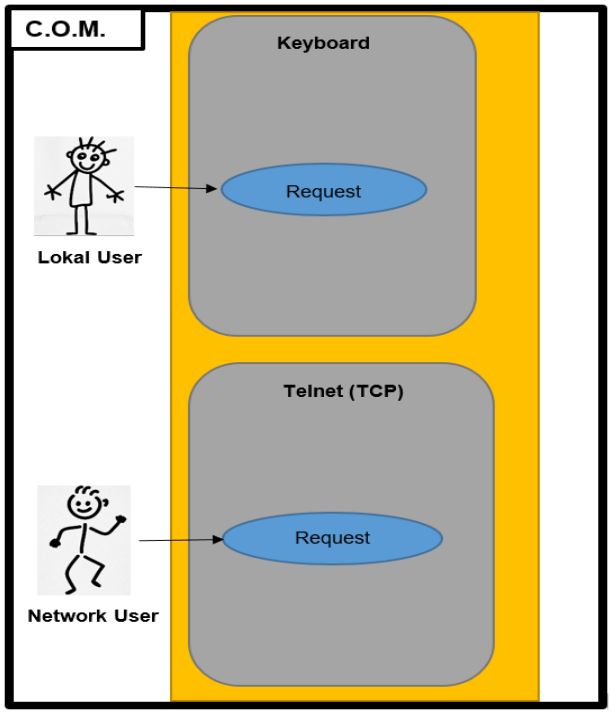

Abbildung 5: UPDATE - Use-Case Diagram Chain Operation Mode

1.3 Sequenzdiagramm

Das Sequenzdiagramm beschreibt ein Szenario im Chain-Mode, in dem ein Paket vom linken Förderband an unser Förderband „MyBelt“ transportiert wird. Nach dem Abfahren des Profils wird das Paket von unserem Förderband „MyBelt“ an das rechte Förderband übergeben.

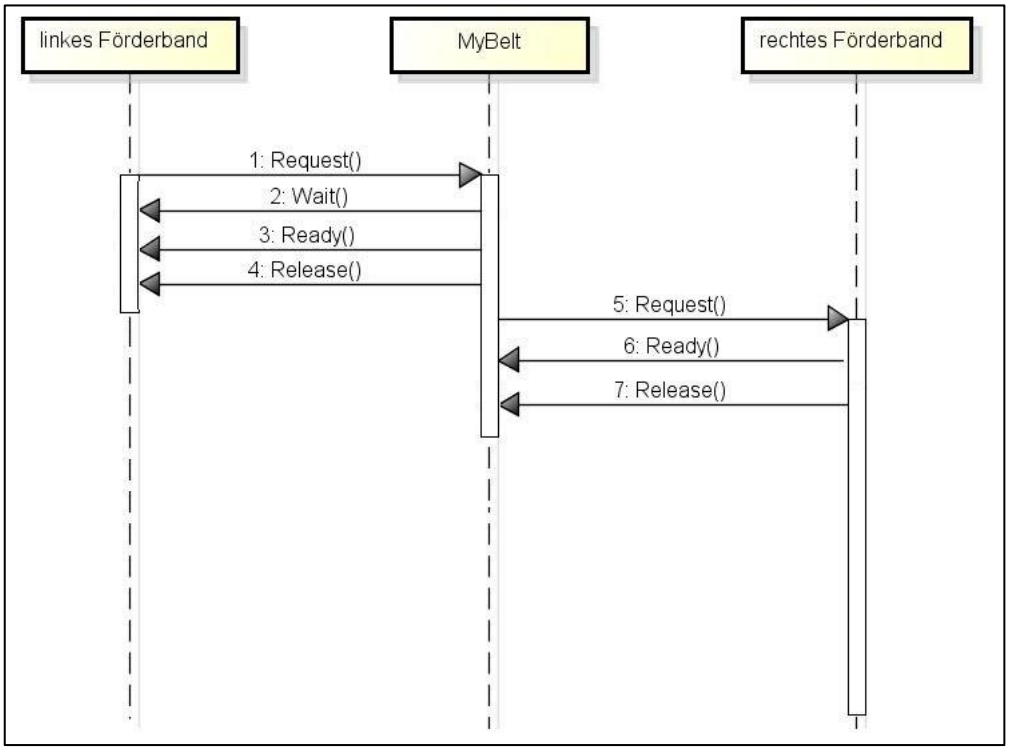

Abbildung 6: Sequenzdiagramm

1.4 Aktivitätsdiagramm

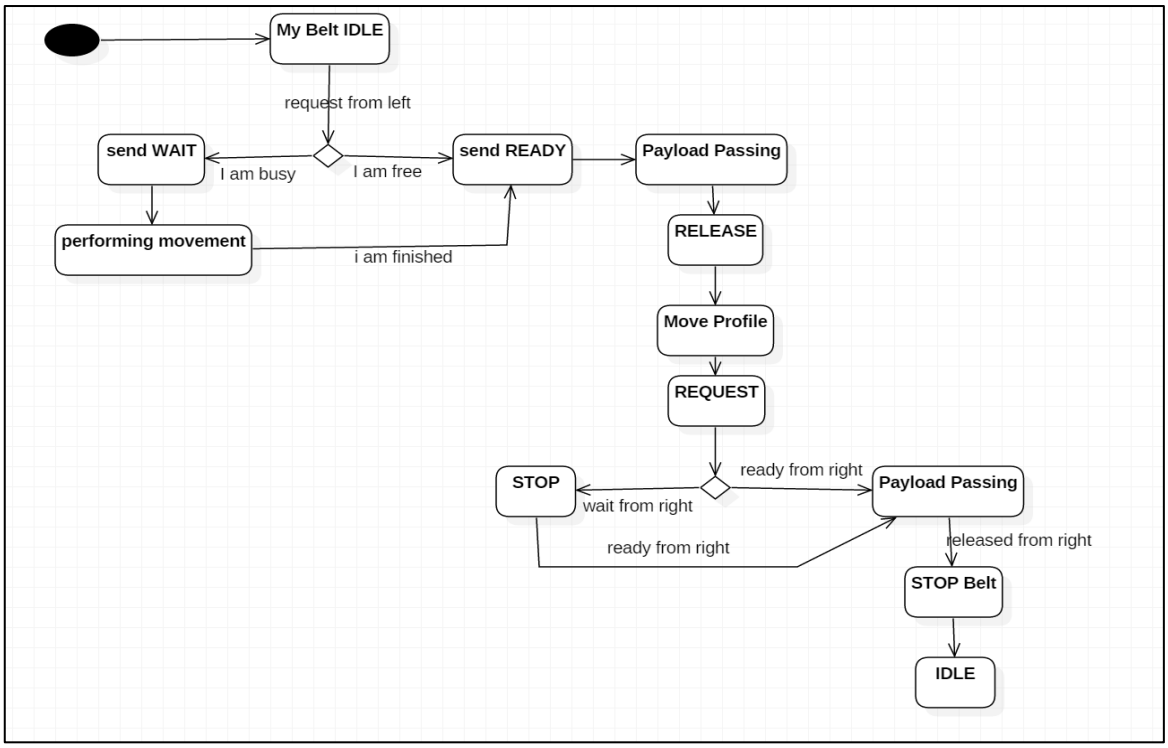

Abbildung 7: Aktivitätsdiagramm

2. UML Klassendiagram

In Abbildung 8 und Abbildung 9 sind die UML-Klassendiagramme vor und nach der Programmierung abgebildet. Der Kern des Klassendiagrams mit dem Namen „MyBelt“ blieb jedoch erhalten. Einige Methoden und Attribute wurden in der Programmierphase verändert und umbenannt.

Eine wesentliche Änderung gab es beim TCP-Server, TCP-Client, Telnet-Server und dem Keyboard. Geplant war im ersten Schritt die Implementierung eines Interfaces, jedoch hatte sich während der Programmierung gezeigt, dass ein Interface „ICommand“ hier keinen großen Mehrwert bringen würde und somit gestrichen wurde. Ebenfalls hatte sich während der Programmierung herausgestellt, dass z.B. durch die IP-Zuweisung des Masters es sinnvoller ist den TCP-Client durch den TCP-Server erzeugen zu lassen.

In der Klasse Motor wurden die „Hardwarefunktionen“ verwendet, um den Motor zu starten und stoppen, sowie die Ist-Drehzahl und die Richtung zu bestimmen. Der Regler greift auf den Motor zu, um die Ist-Drehzahl für die Regelung einzulesen. Die Solldrehzahl wird in der Klasse „MyBelt“ definiert, welche dann die Information an den Regler weitergibt. Die Klasse „Display“ bedient sich ihrer benötigten Informationen ebenfalls aus der Klasse „Motor“.

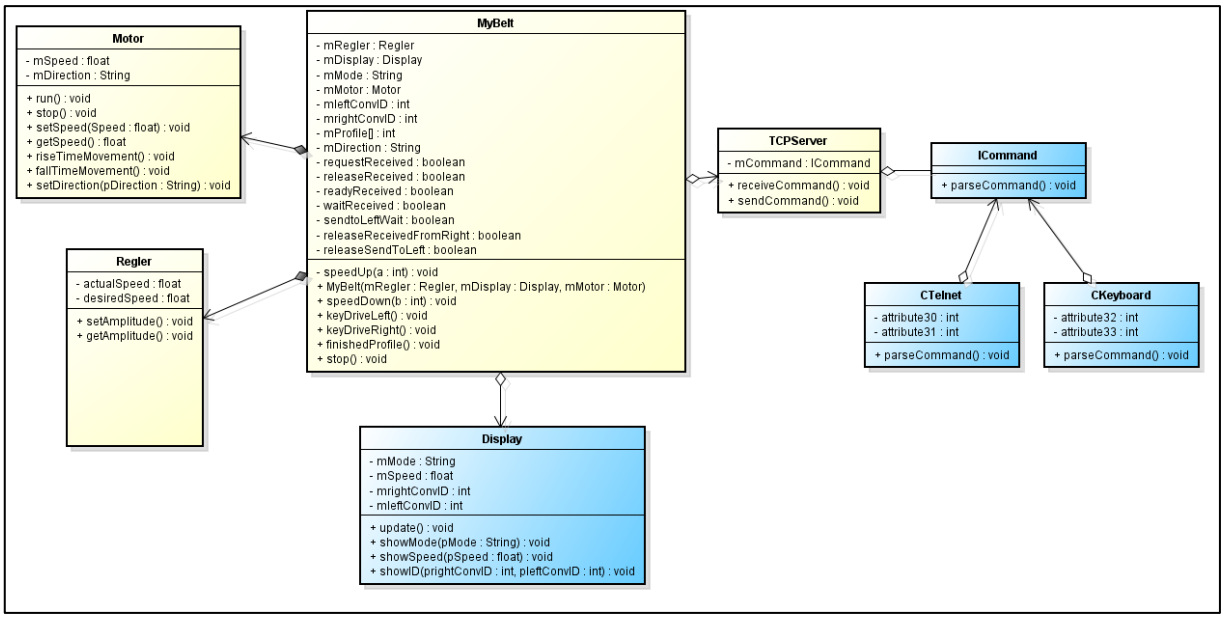

Abbildung 8: UML-Klassendiagramm

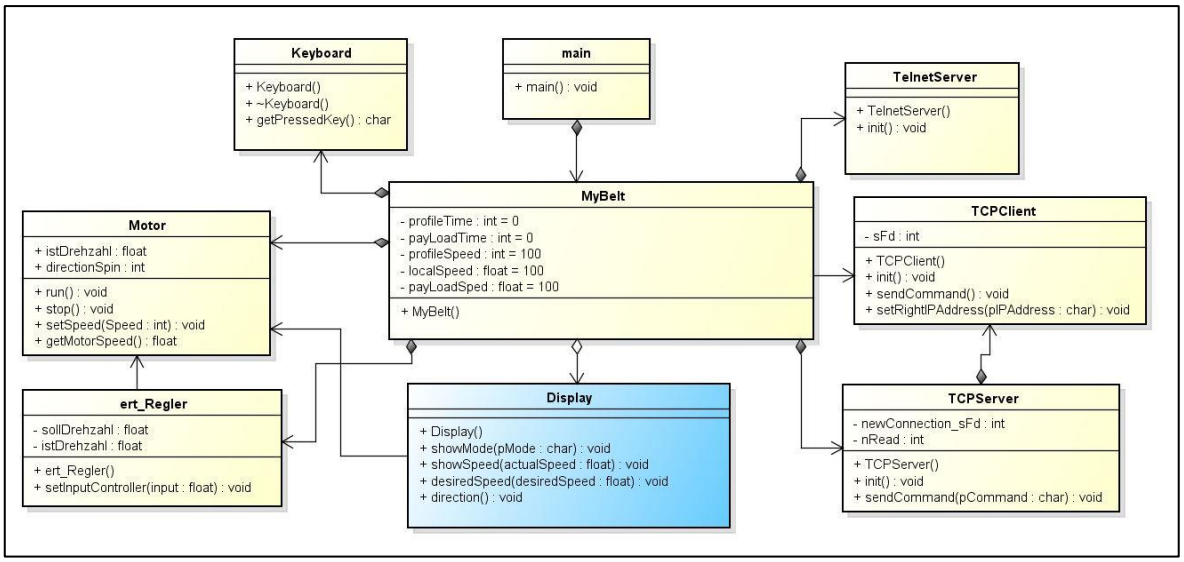

Abbildung 9: UML-Klassendiagramm Nachher

3. Zustandsdiagramme

Folgende Zustandsdiagramme beschreiben das Verhalten des Förderbandes im „Local Sevice Operation Mode“ und im „Chain Operation Mode“.

Erweiterungen - Update:
Folgende Erweiterungen wurden für beide Zustandsautomaten (LOCAL-Mode und CHAIN-Mode) vorgenommen. Mit „HoldLocalMode“ und „HoldChainMode“ werden die zwei Zustandsautomaten initialisiert, um über das Keyboard das Wechseln von einem Zustandsautomaten zum anderen zu ermöglichen.

Für die ständige Abfrage, ob eine Taste gedrückt wird, sowie die Aktualisierung von den Parametern auf dem Display, wird in diesem Projekt ein dritter Zustandsautomat mit einem einzigen Zustand implementiert.

Im „Local Service Operation Mode“ und im „Chain Operation Mode“ wurden die Transitionsnamen mit einer Zahl bzw. Buchstaben vermerkt. Diese stimmen direkt mit dem Keyboard überein, um eine vereinfachte Bedienung zu ermöglichen sowie mittels Tastendruck alle benötigten Transition auch für Testzwecke zu erzeugen.

3.1 Local Service Operation Mode

Im „Local Operation Mode“ haben sich die Zustände vor und nach der Programmierung nicht verändert, es ist lediglich ein Zustand „HoldLocalMode“ wie schon erwähnt hinzugekommen. Zusätzlich wurden die Transitionen aus Bedienerfreundlichkeit und aus Testzwecken angepasst.

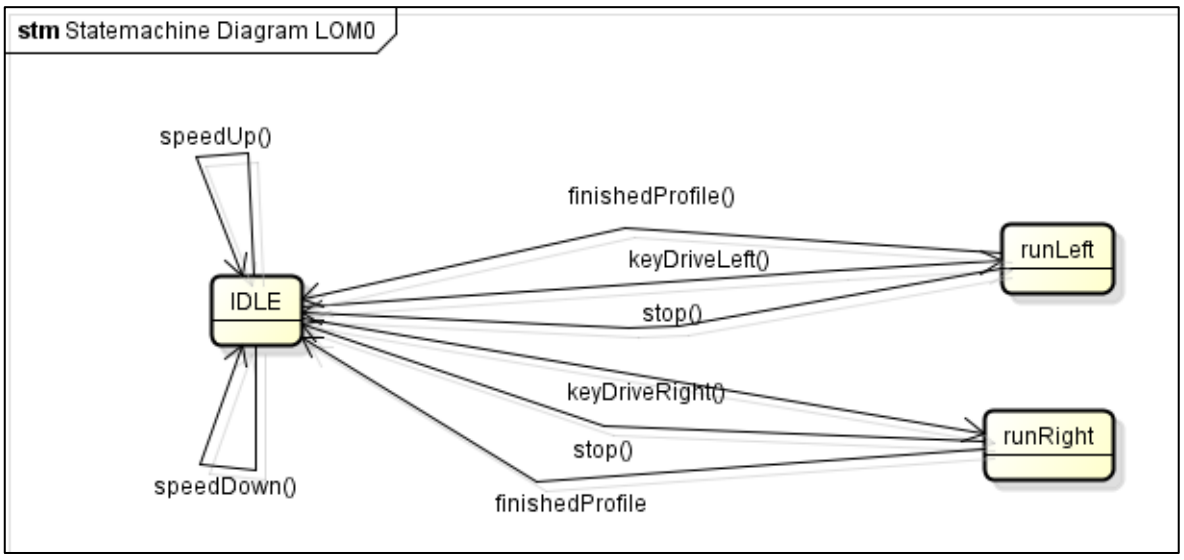

Abbildung 10: Zustandsdiagramm Local Service Operation Mode

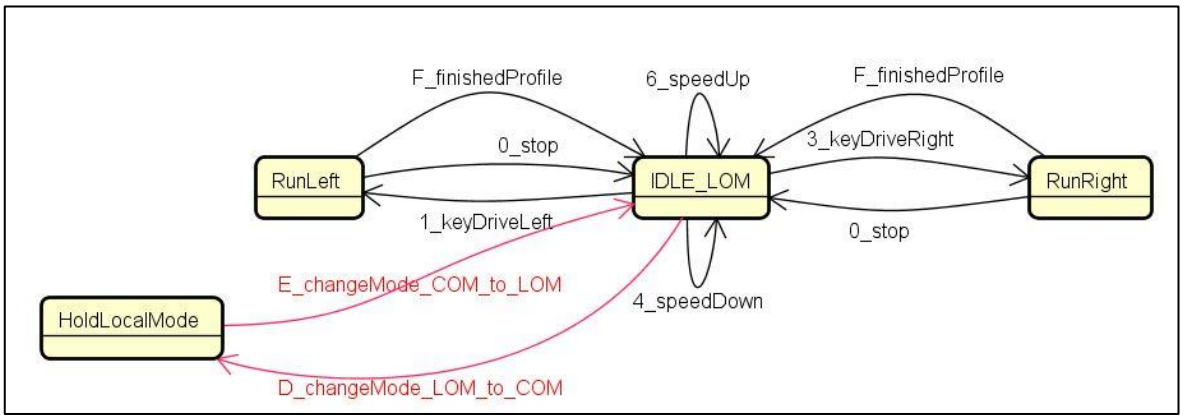

Abbildung 11: Zustandsdiagramm Local Service Operation Mode Nachher

3.2 Chain Operation Mode

Im „Chain Operation Mode“ haben sich die Zustände vor und nach der Programmierung nicht verändert, es ist lediglich ein Zustand „HoldChainMode“ wie schon erwähnt hinzugekommen. Für jegliche erneute Anfrage vom TCP-Server oder TCP-Client wurden beim Abfahren des Profils und des „Payloadpassing“, ein Timer1 mit einer Zeitschlitzbreite von 50ms hinzugefügt. Bei der Transition „Trigger1“ wird für eine Sekunde gewartet, damit der Übergang zu „Moveprofile“ sichtbar ist.

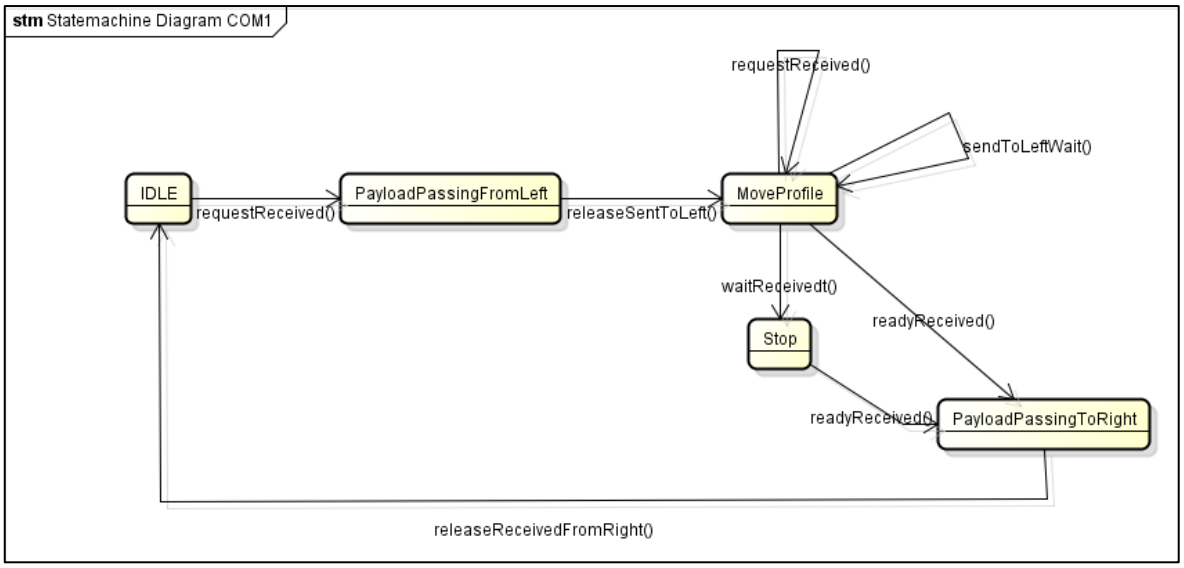

Abbildung 12: Zustandsdiagramm Chain Operation Mode

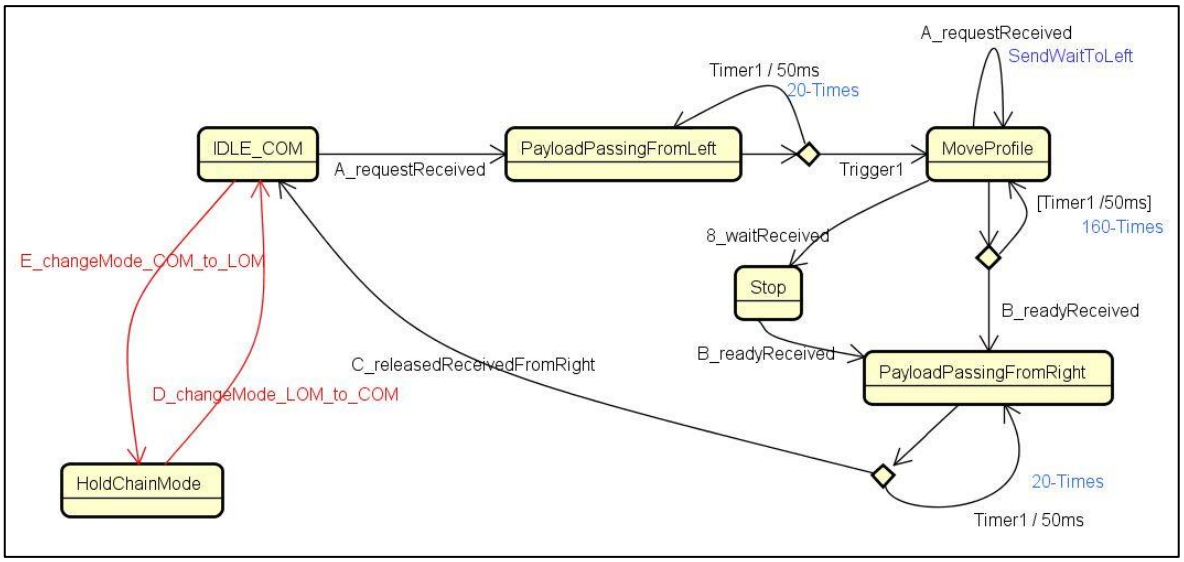

Abbildung 13: Zustandsdiagramm Chain Operation Mode Nachher

4. Closed Loop Control (CLC)

4.1 Kopplung CLC mit Zustandsautomat

Die Strecke des Motors wurde in Simulink modelliert und war gegeben. Die Motordrehzahl wurde mittels eines PI-Regler geregelt und somit die Regelschleife geschlossen, siehe Abbildung 14. Der Input des PI-Reglers ist die Reglerdifferenz, welche sich aus der Differenz von Solldrehzahl minus Motor-Istdrehzahl ergibt. Der Regler-Output ist die Stellgröße, die in diesem konreten Fall die angelegte Ankerspannung ist.

Die gegebenen Modell- und Reglerparameter wurden nicht verändert.

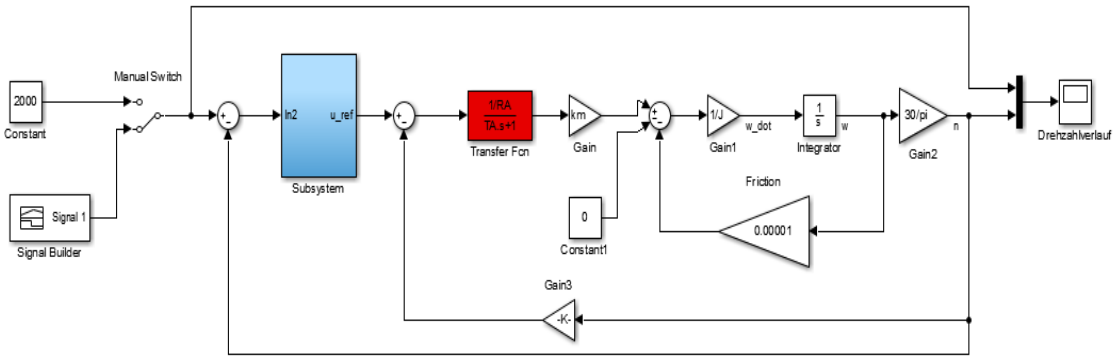

Abbildung 14: Regelkreis mit PI-Regler und zu regelnden Motor als Strecke

Mittels Codegenerierung wurde der Simulinkregler in C-Code übersetzt und in eine C++ Datei eingebunden, damit der Regler im Projekt für die Motorregelung verwendet werden konnte. Hierbei waren folgende Punkte zu beachten:

a) Headerdatei "Subsystem.h" wurde mittels „extern C“ implementiert
b) Im „myBelt“ wird der Konstruktur des Reglers aufgerufen, wodruch dieser einen einen separaten Task für den Regler erzeugt.
c) Dem Regler wurde eine Funktion „setInputController()“ hinzugefügt um den Sollwert der Drehzahlvorgabe aus den „myActions“ vom Zustandsautomaten heraus zu setzen.
d) Im Regler wird ca. alle 15 µs zyklisch die Istdrehzahl des Motors ausgelesen, wodurch für die Drehzahlbestimmung die Anzahl der Encoderimpulse auf die neue Zeitschlitzbreite umgelegt werden musste.
e) Als **Reglerinput** wurde die Solldrehzahl minus der Motor-Istdrehzahl verwendet. Wobei darauf zu achten war, dass die Motor-Istdrehzahl mit der Drehrichtung (+1 / -1) multipliziert werden musste.
f) Der **Regler** hatte als **Output** die Ankerspannung (-18 V … +18 V), welche auf den digitalen Bereich von 0…4095 umgelegt werden musste. Hierbei entsprach zum einen 1 Volt = 220 digitalen Schritten und zum anderen musste die 0V Ankerspannung auf die 0 rpm Drehzahlvorgabe umgelegt werden was einem

digitalen Wert von 2580 entsprach. Zusätzlich musste darauf geachtet werden, dass eine positive Ankerspannung ein Rechtsdrehen des Motors bewirkt.

g) Für die Debugg-Ausgaben musste darauf geachtet werden, dass eine „printf()“-Funktion nicht mit mehr als 15 Zeichen überlastet wird, da sonst die Reglerzykluszeit überschritten wird.

4.2 Regelgüte – Sprungantwort

Die Qualität des Reglers wurde am realen System, sprich dem realen DC-Motor, verifiziert. Hierzu wurde dem PI-Regler ein Sollwertsprung von 0 rpm auf 1000 rpm vorgegeben und die Sprungantwort des geschlossenen Regelkreises anhand des Drehzahlverlaufs des Motos aufgezeichnet (siehe Abbildung 15).

Folgende Zeiten wurden in der Sprungantwort erreicht:

- Risetime: $T_r = 0{,}32\ ms$ → gefordert aus Requirements: $T_{r\_require} = 5\ ms$
- Settletime: $T_s = 0{,}65\ ms$ → gefordert aus Requirements: $T_{s\_require} = 15\ ms$

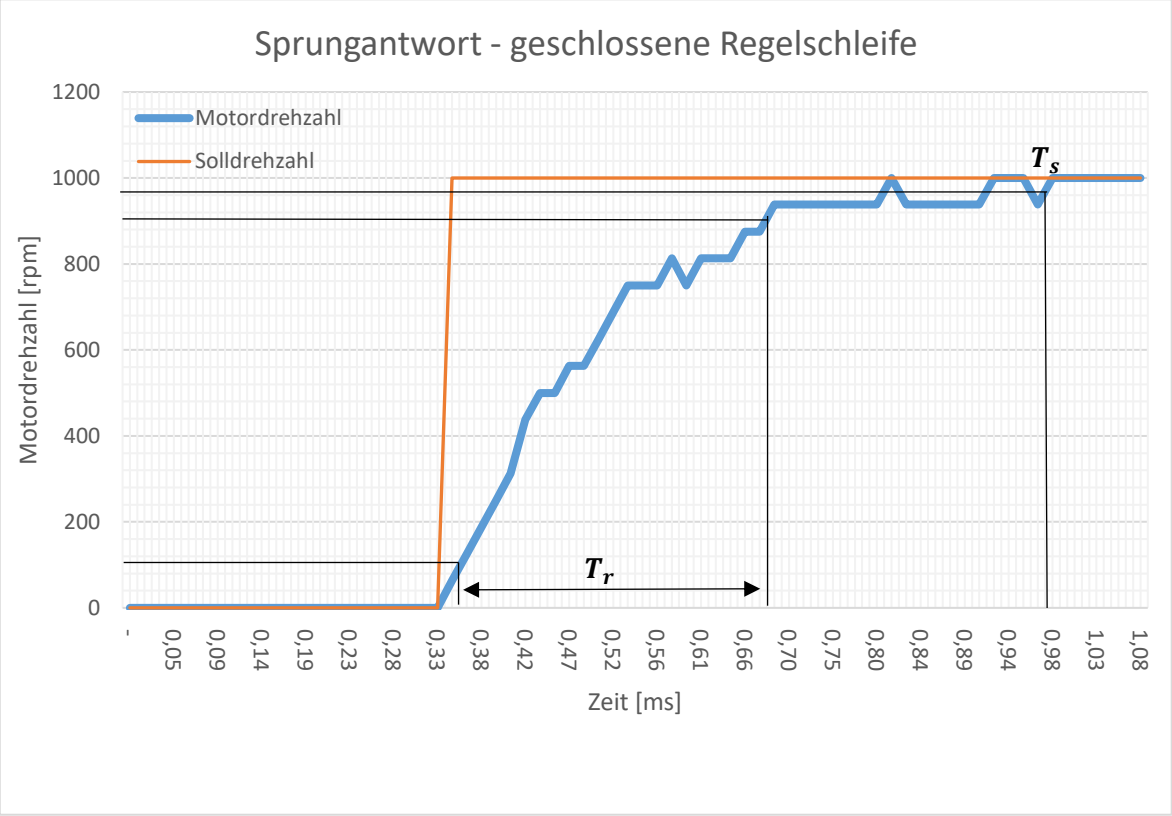

Abbildung 15: Sprungantwort der geschlossenen Reglerstrecke mit realem DC-Motor

Die in den Requirements geforderten Systemzeiten der Sprungantwort, bezogen auf die Risetime T_r und die Settletime T_s, wurden somit eingehalten und erfüllt. In der Sprungantwort besteht kein Überschwingen, da die Sprungantwort ein PT1-Verhalten aufweist. Hierdurch wird die Peaktime sowie die Anforderung an das Überschwingen (c_{max}) irrelevant und somit nicht verletzt.

Die Anforderung zur Regelabweichung wird ebenfalls eingehalten wie aus Abbildung 15 ersichtlich. Der Integratoranteil des Reglers sorgt dafür, dass die Regelabweichung gegen 0 geht. Der Regler könnte durch den P-Anteil noch schneller eingestellt werden, wodurch die Risetime weiter verringert werden könnte. Hierdurch würde aber ab einer gewissen Verstärkung des P-Anteils ein Überschwingen in der Sprungantwort entstehen, welche zu überprüfen wäre.

Generell ist noch zu erwähnen, dass ein DC-Motor ein PT2-Verhalten aufweist, welches durch eine mechanische und eine elektrische Zeitkonstante auftritt. Diese beiden Zeitkonstanten liegen physikalisch bedingt um mindestens eine Zehnerpotenz voneinander entfernt. Dies ist auch der Grund, weshalb bei einem DC-Motor die mechanische Zeitkonstante die dominierende Zeitkonstante für das Systemverhalten darstellt, welche regelungstechnisch betrachtet werden muss. Dies ist auch der Grund warum ein DC-Motor mit einem PT2-Verhalten mit zwei Zeitkonstanten durch ein PT1-Verhalten mit nur einer Zeitkonstanten modelliert und geregelt werden kann.