Inbetriebnahme der Fischertechnik-9V-Modellfabrik mit Schwerpunkt Backup und Programmierung

T3_3101 Studienarbeit

für das

5. und 6. Theoriesemester

des Studiengangs Informatik

an der Dualen Hochschule Baden-Württemberg Stuttgart Campus Horb

Joel Schmid

Mai 2020

Bearbeitungszeitraum	2 Semester
Matrikelnummer, Kurs	2682198, TINF2017
Betreuer	Prof. Dr.-Ing. Olaf Herden

Erklärung

gemäß § 5 (3) der „Studien- und Prüfungsordnung DHBW Technik“
vom 29. September 2015.
Ich versichere hiermit, dass ich meine T3_3101 Studienarbeit mit dem Thema

Inbetriebnahme der Fischertechnik-9V-Modellfabrik mit Schwerpunkt Backup und Programmierung

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.
Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Ort, Datum	Unterschrift

Zusammenfassung

In der vorliegenden Studienarbeit wird die Inbetriebnahme der fischertechnik Modellfabrik behandelt. Dies beinhaltet einen Vergleich der verschiedenen Programmiersprachen für den TXT-Controller zu und die Auswahl eine geeignete Sprache für die Programmierung der Modellfabrik sowie für Laborübungen. Nach Gegenüberstellung von Vor- und Nachteilen der einzelnen Programmiersprachen wird C/C++ für die Programmierung des TXT-Controllers ausgewählt. Um die weitere Entwicklung von Steuerungsprogrammen für den TXT-Controller zu vereinfachen, werden über die Programmierschnittstelle von fischertechnik weitere Abstraktionsebenen entworfen. Auf diese Abstraktionsebenen aufbauend wird ein Beispielprogramm für den Showbetrieb der Modellfabrik entwickelt. Abschließend wird neben der Ausarbeitung von zwei Laborübungen zur Programmierung des TXT-Controllers auch ein Backupkonzept für die Programme des Showbetriebs der Modellfabrik vorgestellt.

Hinweise

In dieser Arbeit wird aus Gründen der besseren Lesbarkeit das generische Maskulinum verwendet. Weibliche und anderweitige Geschlechteridentitäten werden dabei ausdrücklich mitgemeint, soweit es für die Aussage erforderlich ist.

Diese Studienarbeit entstand parallel zur Studienarbeit von Johannes Gaiser mit dem Thema **Inbetriebnahme der Fischertechnik-9V-Modellfabrik mit Schwerpunkt Dokumentation und Prozessvisualisierung**. An einigen Stellen überschneidet sich der Inhalt dieser Arbeiten oder es werden auf Kapitel in der jeweils anderen Arbeit verwiesen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Tabellenverzeichnis

Abkürzungsverzeichnis

API Application Programming Interface

CPU Central Processing Unit

DLL Dynamic Link Library

IDE Integrated Development Environment

LLWin Lucky Logic for Windows

NTC Negative Temperature Coefficient

PWM Pulsweitenmodulation

SSH Secure Shell

1 Einleitung

Das erste Kapitel dieser Studienarbeit behandelt die Motivation und Problemstellung, sowie Vorgehen und Ziel dieser Arbeit.

1.1 Motivation und Problemstellung

Baukästen von fischertechnik finden ihren Einsatz nicht nur zuhause in Kinderzimmern und Bastelwerkstätten, auch in industriellem, schulischem und akademischem Umfeld sind Modelle von fischertechnik zu Simulations-, Forschungs- und Lehrzwecken anzutreffen [fisb].

Für den produktiven Einsatz zu Lehrzwecken hat die DHBW Stuttgart Campus Horb ein betriebsbereites Trainingsmodell, die fischertechnik-9V-Modellfabrik[1] gekauft. Diese Modellfabrik soll neben dem Einsatz im regulären Studienbetrieb zu Lehrzwecken auch für den Showbetrieb eingesetzt werden können. Bei Veranstaltungen wie dem Girls-day oder dem Studieninformationstag soll die Modellfabrik als Eyecatcher dienen und Interesse an Themen wie Programmierung, Automatisierung und Industrie 4.0 wecken.

Bevor die Modellfabrik für den produktiven Einsatz in Lehrbetrieb oder Showbetrieb genutzt werden kann, muss sie in Betrieb genommen werden. In dieser Studienarbeit wird die Inbetriebnahme der Modellfabrik mit dem Schwerpunkt Programmierung und Backupkonzept behandelt.

[1] Website: `https://www.fischertechnik.de/de-de/produkte/lehren/trainingsmodelle/536629-edu-fabrik-simulation-9v-education`

1.2 Vorgehen und Ziel der Arbeit

Das Vorgehen in dieser Arbeit kann in fünf unterschiedliche Bereiche eingeteilt werden.

Zuerst werden in Kapitel 2 allgemeine Grundbegriffe erläutert sowie die historische Entwicklung der programmierbaren fischertechnik Controller dargestellt. In diesem Zusammenhang wird auch eine Übersicht über die in dieser Arbeit behandelte Modellfabrik gegeben.

Daran anschließend folgt in Kapitel 3 eine Analyse der für den TXT-Controller verfügbaren Programmiersprachen. Das Ziel in diesem Teilabschnitt ist eine aussagekräftige Gegenüberstellung der Programmiersprachen mit ihren Vor- und Nachteilen, sowie die Auswahl einer Sprache für die Übungen im Lehrbetrieb und der Fabriksteuerung im Showbetrieb.

Im dritten Schritt wird die Steuerung der Modellfabrik für den Showbetrieb umgesetzt und Implementiert. Dies beinhaltet neben der eigentlichen Implementierung der Fabriksteuerung in Kapitel 5 auch den Entwurf einer Entwicklerfreundlichen API in Kapitel 4 als Vorstufe und zur Vereinfachung der eigentlichen Entwicklung der Fabriksteuerung.

Als letzter großer Teilabschnitt wird in Kapitel 6 dieser Arbeit die Nutzbarkeit der Modellfabrik für den Lehrbetrieb verbessert. Dies wird durch Ausarbeitung eines Backupkonzeptes zum Wechsel zwischen Lehr- und Showbetrieb sowie dem Entwurf zweier Laborübungen auf Basis der ausgewählten Programmiersprache erreicht.

Abschließend werden in Kapitel 7 die wichtigsten Punkte dieser Arbeit in einem Fazit zusammengefasst.

2 Grundlagen

In diesem Kapitel werden die notwendigen Grundlagen für das weitere Verständnis der Arbeit vermittelt. Zuerst wird auf allgemeine Begrifflichkeiten eingegangen. Darauf folgt der fischertechnik-spezifische Teil, indem die Entwicklung der programmierbaren Controller über die einzelnen Produktgenerationen hinweg dargestellt sowie eine Übersicht über die Modellfabrik gegeben wird.

2.1 Cross Compiler

Ein Cross Compiler ist ein Compiler, der auf einer Hostplattform A ein ausführbares Programm für eine Zielplattform B baut. Die Plattformen A und B können sich dabei in Betriebssystem, Central Processing Unit (CPU) und Ausführungsformat unterscheiden [Fou20a].

Cross Compiler werden häufig für die Entwicklung von Programmen für Eingebettete Systeme mit sehr geringen verfügbaren Ressourcen genutzt, da diese nicht ausreichend sind, um selbst den Compiler auszuführen.

2.2 Secure Shell

Secure Shell (SSH) ist ein Netzwerkprotokoll, welches die Möglichkeit eines sicheren Einloggens, sowie Dateiübertragungen von einem Computer zu einem anderen Computer über das Internet, oder andere nicht vertrauenswürdige Netzwerke ermöglicht [Ylö96]. Standardmäßig nutzt SSH TCP/IP als Transportprotokoll.

Nach Eingabe der Anmeldedaten durch den Nutzer wird der Verbindungsaufbau zwischen Client und Server nach dem in Abbildung 2.1 dargestellten Vorgehen aufgebaut.

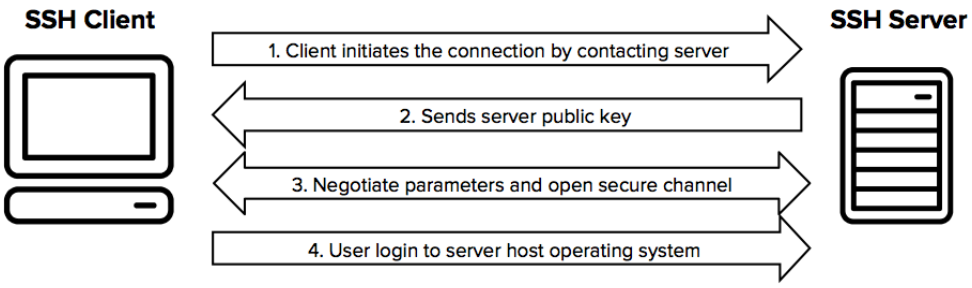


Abbildung 2.1: Verbindungsaufbau zu einem SSH Server

https://images.ctfassets.net/0lvk5dbamxpi/7AWtdnEb8L92WkLkP4Tri8/23c91236721ec34ba42635ea5be05f2c/SSH_simplified_protocol_diagram

Nachdem die Verbindung zum SSH Server aufgebaut ist, können auf diesem z.B. Remotekommandos im Terminal ausgeführt werden.

2.3 Make

Make ist ein Softwarewerkzeug, welches die Übersetzung von Quellcode in ausführbaren Maschinencode koordiniert. Make bezieht das Wissen, wie eine ausführbare Datei gebaut werden soll aus dem sogenannten *Makefile*. In diesem sind *Targets* (Ziele), deren Abhängigkeiten von Quelldateien, sowie Regeln, wie die Ziele gebaut werden sollen, wenn sich eine Quelldatei ändert [Fou20b].

```
1 Ampel.o : src/Ampel.cpp
2   $(CXX) $(CXXFLAGS) $^ -o obj/$@
```

Bei Änderungen an der C++ Quelldatei `Ampel.cpp` wird das Ziel `Ampel.o` vom C++ Compiler mit dem Aufruf `$(CXX) $(CXXFLAGS) $^ -o obj/$@` neu gebaut. Dabei sind in den Variablen `$(CXX)` und `$(CXXFLAGS)` der Compilerpfad und die Compilerflags hinterlegt. Das Wildcard `$^` verweist auf die in der Abhängigkeitsregel hinterlegte C++ Quelle (können auch mehrere sein). Mit dem Wildcard `$@` wird dem Compiler das gewünschte Ziel *Ampel.o* mitgeteilt. Um mit dem vorliegenden Makefile aus der Quelldatei `Ampel.cpp` die Zieldatei `Ampel.o` zu bauen, ist ein einfacher Kommandozeilenbefehl mit dem Aufruf von Make notwendig: `make Ampel.o`[1]. Das Working-Directory des Terminals muss das Verzeichnis

[1] Für die Windows Portierung von Make über MinGW ist der Aufruf: `mingw32-make Ampel.o`

sein, welches das Makefile enthält.

2.4 Pulsweitenmodulation

Pulsweitenmodulation (PWM) ist ein Verfahren, mit dem durch hochfrequentes Ein- und Ausschalten digitaler Signale eine im Vergleich zur Ausgangsspannung niedrigere Spannung erzeugt wird. Das Verhältnis zwischen Einschaltzeit t_{ein} und Periodendauer T des Signals wird als Tastverhältnis p oder Tastgrad (engl. Duty Cycle) bezeichnet[mik]. Es gilt:

$$p = \frac{t_{ein}}{t_{ein} + t_{aus}} = \frac{t_{ein}}{T}$$

Dies kann grafisch wie in Abbildung 2.2 dargestellt werden.

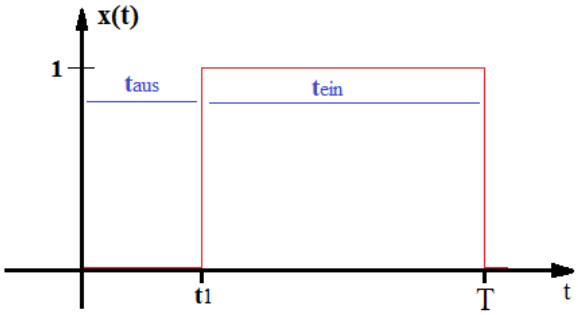


Abbildung 2.2: Verhältnis zwischen Einschaltzeit und Ausschaltziet

`https://www.mikrocontroller.net/wikifiles/c/c6/Pwmdoc.png`

Die aus einer solchen Rechteckspannung resultierende zeitliche Mittelwertspannung (simulierte Analogspannung) U_m berechnet sich durch [mik]:

$$U_m = V_{CC} \cdot p = V_{CC} \cdot \frac{t_{ein}}{t_{ein} + t_{aus}}$$

Bei einer Lampe, die mit einem Tastverhältnis von 0,5 angesteuert wird, ist die empfundene Helligkeit halb so groß.

2.5 Entwicklung der programmierbaren fischertechnik Controller

In den folgenden Abschnitten werden die einzelnen Generationen der fischertechnik Controller näher vorgestellt und mit dem jeweiligen Vorgängermodell verglichen. Der ROBO LT Controller ist in diesem Vergleich nicht aufgeführt, da er als Einsteigermodell nur fischertechnik-Modelle mit wenigen Anschlüssen ansteuern kann.

2.5.1 Computing Interface

fischertechnik bietet bereits seit dem Jahr 1985 die ersten Interfaces an, mit denen die fischertechnik-Modelle über einen Rechner im Onlinebetrieb gesteuert werden konnten [coma]. Die unterschiedlichen Interfaces konnten von jeweils einem der damals verbreiteten Rechner (C64, Apple II, ...) angesteuert werden [Mül05].

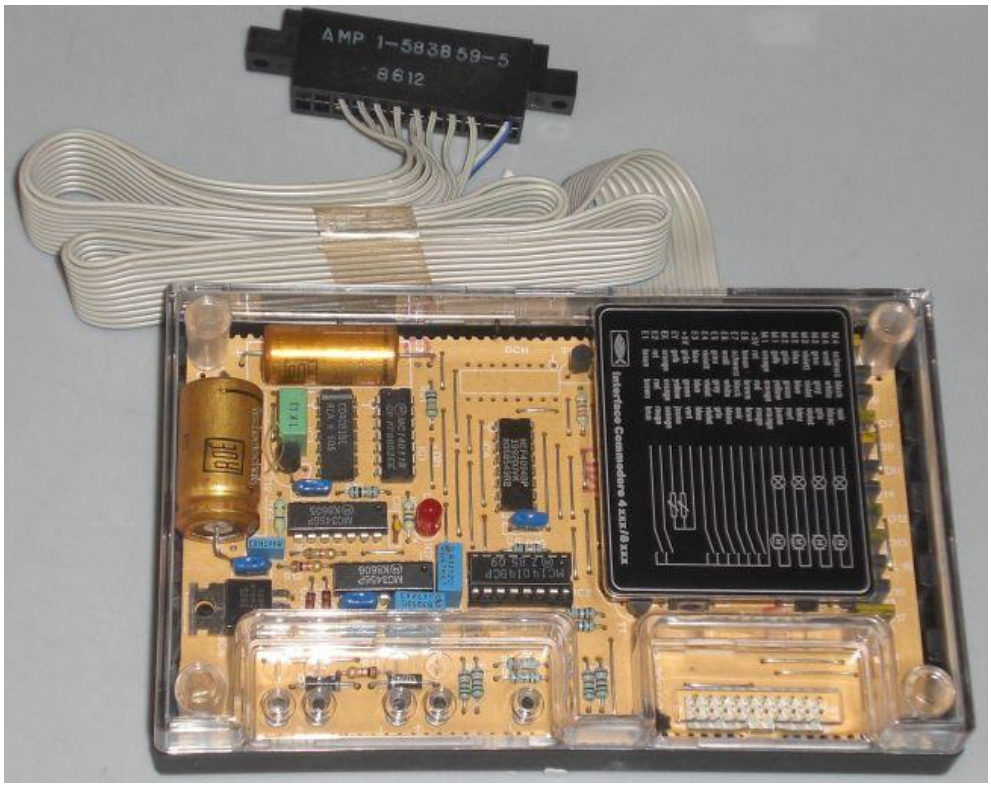


Abbildung 2.3: Computing Interface

https://ftcommunity.de/knowhow/computing/computing_interfaces/30561.png

2.5.2 Intelligent Interface

Im Jahr 1997 stellt fischertechnik mit dem Intelligent Interface die Nachfolgergeneration zum Computing Interface vor. Mit der neuen Generation ist neben dem Onlinebetrieb erstmals auch ein Downloadbetrieb möglich [fisa]. Dadurch ist es jetzt möglich, mobile Roboter mit größerer Bewegungsfreiheit zu entwickeln, da zum Betrieb keine Rechnerverbindung mehr benötigt wird. Das Intelligent Interface verfügt über 4 Motorausgänge sowie über 8 digitale und 2 analoge Eingänge. Programmiert wird das Intelligent Interface mit der grafischen Entwicklungsumgebung Lucky Logic for Windows (LLWin).

fischertechnik hat für das Intelligent Interface mehrere Dynamic Link Librarys (DLLs) bereitgestellt, mit deren Hilfe das Intelligent Interface und dessen Nachfolgermodell ROBO Interface in diversen Sprachen programmiert werden kann [Mül11a] (teilweise auch ROBO TX). Unter den unterstützten Sprachen finden sich neben Python, Pascal und Perl auch Java und C#, sowie einige weitere.

2.5.3 ROBO Interface

2004 stellt fischertechnik mit dem Robo Interface die nächste Generation der programmierbaren Controller vor. Es besitzt genau wie sein Vorgängermodell 8 digitale Eingänge und 4 Motorausgänge, sowie je zwei Eingänge für analoge Spannungsmessung, für analoge Widerstandsmessung und Abstandsmessung [Mül11b]. Die ansteuerbaren Anschlüsse des ROBO Interface können mit bis zu drei ROBO I/O Extensionmodulen erweitert werden.

Genau wie auch die nachfolgenden Generationen können Programme für das ROBO Interface mit der grafischen Programmierumgebung ROBOPro entwickelt werden. Näheres zu ROBOPro zu einem späteren Zeitpunkt in Abschnitt 3.2. Für die Programmierung im Onlinebetrieb können ebenfalls die im vorigen Abschnitt genannten DLLs verwendet werden.

Abbildung 2.4: ROBO Interface

https://www.ftcommunity.de/ftComputingFinis/images/ROBO480.jpg

2.5.4 ROBO TX

Der ROBO TX Controller ist die vierte Generation der fischertechnik Controller. Als erster fischertechnik Controller besitzt der ROBO TX ein eigenes Display, auf dem Texte und einfache Grafiken in schwarz/weiß dargestellt werden können. Das Display hat eine Auflösung von 128x64 Pixel. Als weitere Schnittstellen bietet der ROBO TX Controller Bluetooth 2.0, I2C und RS485 [comb]. Der ROBO TX Controller verfügt im Gegensatz zu seinem Vorgänger ROBO Interface über 8 Universaleingänge, die sowohl analoge als auch digitale Signale auswerten können. Zusätzlich zu den 4 Motorausgängen besitzt er 4 schnelle Zähleingänge mit einer Frequenz von 1 kHz für die Ansteuerung von Schrittmotoren [Mül09].

Abbildung 2.5: ROBO TX Controller

https://ftcommunity.de/knowhow/computing/tx/TXController.jpg

2.5.5 ROBOTICS TXT

Der ROBOTICS TXT-Controller, im weiteren Verlauf der Arbeit nur noch TXT oder TXT-Controller genannt, bildet die aktuelle Steuerungsgeneration für fischertechnik-Modelle und bringt einige Neuerungen im Vergleich zur Vorgängerversion mit. Erstmals kommt in dieser Generation als Firmware ein Linux-Betriebssystem zum Einsatz, welches mit dem ebenfalls neuen 3"touchfähigen Farbdisplay bedient werden kann. Mit dem eingebauten WLAN-Modul kann jetzt auch auf Ressourcen im Internet zugegriffen werden [comc].

Zu den Ein- und Ausgängen wurde ein USB-Port hinzugefügt, über den die fischertechnik Kamera an den TXT-Controller angeschlossen und genutzt werden kann. An den 'normalen' IO-Pins zur Ansteuerung von Modellen hat sich bei diesem Generationenwechsel nichts geändert.

Im Gegensatz zum Vorgängermodell hat der TXT jedoch nur noch einen Anschluss zur Kommunikation mit anderen TXT-Controllern. Somit können jetzt nur noch zwei TXT über Kabel miteinander kommunizieren.

Abbildung 2.6: ROBOTICS TXT Controller

`https://ftcommunity.de/knowhow/computing/txt/522429_TXT_Controller-Bildquelle-fischertechnik.jpg`

2.6 Übersicht über die Modellfabrik

Zum Abschluss dieses Kapitels wird die in dieser Arbeit behandelte Modellfabrik vorgestellt. Die Modellfabrik besteht aus drei unabhängigen Einheiten, bei denen jeweils die aktuelle Steuerungsgeneration, der TXT-Controller zum Einsatz kommt:

- Sortiereinheit mit Farberkennung (1 TXT-Controller)
- 3-Achs Vakuumroboter mit Hochregallager (2 TXT-Controller)
- Bearbeitungsstation mit Ofen (2 TXT-Controller)

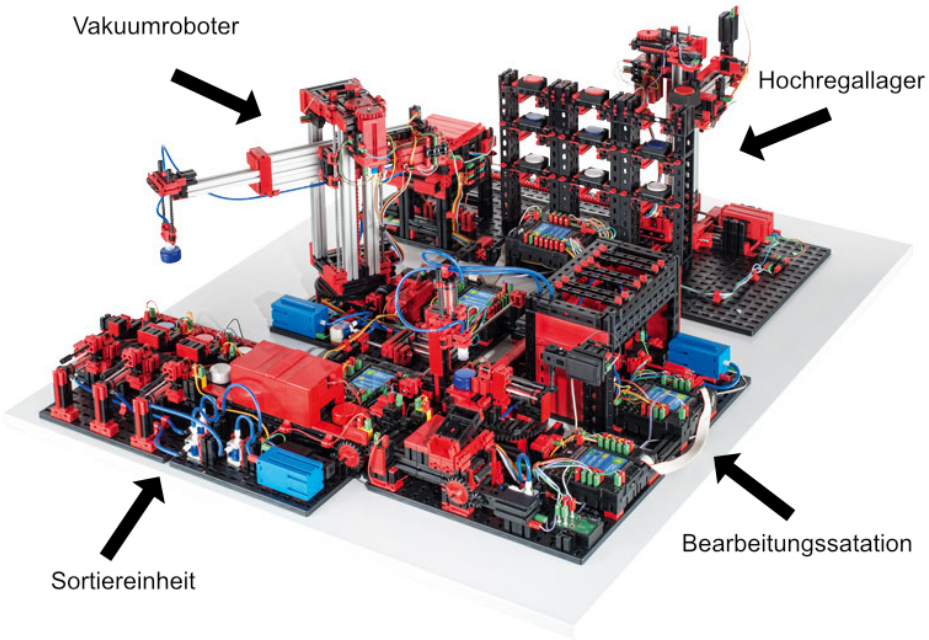


Abbildung 2.7: fischertechnik Modellfabrik

`https://content.ugfischer.com/cbfiles/fischer/produktbilder/ft/FT_Fabrik_Simulation_9V.jpg`

In der Modellfabrik haben die Werkstücke einen zyklischen Durchlauf. Dieser Durchlauf beginnt an der Sortiereinheit der Modellfabrik. Während die Werkstücke auf dem Fließband der Sortiereinheit transportiert werden, erkennt ein Farbsensor ihre Farbe. Anschließend werden die Werkstücke je nach erkannter Farbe in unterschiedliche Zwischenlager einsortiert.

Sobald ein Werkstück von der Sortieranlage in eines der Zwischenlager geschoben wurde, holt der Vakuumroboter dieses Werkstück und stellt es dem Roboter des Hochregallagers bereit. Der Hochregallager-Roboter lagert das vom Vakuumroboter bereitgestellte Werstück ein. Das Hochregallager und der Vakuumroboter können Werkstücke auch auslagern und diese zur Bearbeitungsstation transportieren.

In der Bearbeitungsstation werden die Werkstücke im Ofen erhitzt (durch eine Lampe simuliert). Als zweiter Schritt wird mit einem Zahnrad eine Sägebearbeitung simuliert. Nach der Bearbeitung wird das Werkstück wieder an die Sortiereinheit übergeben.

Für eine detailliertere Beschreibung des Ablaufes, sowie einer Analyse von Aufbau und festgestellten Problemen der Modellfabrik, sei auf die Studienarbeit von Johannes Gaiser verwiesen [Gai20].

3 Programmierung des TXT-Controller

In diesem Kapitel wird die Programmierung des TXT-Controllers näher behandelt. Es existieren verschiedene Sprachen und Bibliotheken, die die Entwicklung von Steuerungen für den TXT-Controller unterstützen.

Es wird die Programmierung des TXT-Controller mit den grafischen Programmiersprachen ROBOPro und Scratch, sowie mit den Hochsprachen C/C++ und Python behandelt. Zum Abschluss dieses Kapitels werden die Ergebnisse miteinander verglichen.

Für eine bessere Vergleichbarkeit der Programmiersprachen, wird bei jeder ein beispielhaftes Ampelsteuerungsprogramm behandelt. Dabei läuft in einer Endlosschleife der übliche Ampelzyklus ab: rot - rot, gelb - grün - gelb - rot.

3.1 Vorbereitungen

Bevor die eigentliche Programmierung des TXT-Controllers beginnen kann, müssen einige Vorbereitungen getroffen werden, um Programmdateien auf den TXT-Controller hochladen zu können.

3.1.1 Verbindung zur Webschnittstelle

Seit der Firmware Version 4.6.6.0 gibt es auf dem TXT-Controller eine Webschnittstelle, über die Dateien auf den TXT-Controller hoch- und heruntergeladen werden können. Diese muss, falls nicht bereits geschehen, in den Einstellungen des TXT-Controller aktiviert werden. Der Eintrag ist unter *Settings -> Security* zu finden. Zur Aktivierung der Webschnittstelle muss die Option *WEB Server* eingeschaltet sein.

Eine Kommunikationverbindung zum TXT-Controller kann auf drei verschiedenen Wegen erreicht werden. Einerseits unterstützt er USB als Kabelbasierte Verbindungsart und andererseits ist er mit Bluetooth und WLAN über Funksignale erreichbar. Egal welche der drei Optionen gewählt wird, ist der TXT-Controller am Rechner immer als Netzwerkgerät zu finden und kann deshalb vom Rechner über die IP-Adresse des Controllers erreicht werden. Je nach gewählter Verbindungsart unterscheidet sich diese:

- USB: 192.168.7.2
- Bluetooth: 192.168.9.2
- WLAN: 192.168.8.2, wenn man mit dem Accesspoint des TXT-Controller verbunden ist, bzw. die vom Router zugewiesene IP-Adresse, wenn der TXT-Controller als Client mit einem Accesspoint verbunden ist

Um auf die Webschnittstelle zugreiffen zu können und auf diese Weise Dateien auf den TXT-Controller laden zu können, muss die IP-Adresse im Adressfeld eines Browsers eingegeben werden. Zur Authentifizierung an der Webschnittstelle wird ein Benutzername und ein Passwort gefordert. Der Benutzername lautet für jeden TXT-Controller *TXT*, das Passwort ist für jeden TXT-Controller individuell. In der Statusleiste im Display des TXT-Controller ist eine Seriennummer (bspw. TXT-3307) eingetragen. Das Passwort besteht aus der darin enthaltenen Zahl, in diesem Fall also *3307*.

3.1.2 SSH Verbindung zum TXT-Controller

Damit eine SSH Verbindung von einem Rechner zum TXT-Controller hergestellt werden kann, muss zuvor auf dem TXT-Controller der SSH Deamon eingeschalten werden. Dazu navigiert man auf dem TXT-Controller nach *Settings -> Security* und kann dort den *SSH Daemon* mit einem Schalter aktivieren.

Anschließend kann mit einem SSH Tool, beispielsweise PuTTY[1], eine Verbindung zum TXT-Controller aufgebaut werden. Erreichbar ist der TXT-Controller unter der im vorigen Abschnitt beschriebenen IP-Adresse. Die Logindaten unterscheiden sich von denen zur Webschnittstelle und sind für jeden TXT-Controller identisch:

- Benutzername: *ROBOPro*

[1] Download: `https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html`

- Passwort: *ROBOPro*

Nach dem einloggen ist man als Standardbenutzer (nicht root) angemeldet und kann sich im Terminal wie auf einem normalen Linux-Betriebssystem verhalten.

3.2 ROBOPro

ROBOPro ist die hauseigene Entwicklungsumgebung von fischertechnik für die Entwicklung von Steuerungsprogrammen für den TXT-Controller, sowie dessen Vorgängermodelle ROBO Interface und ROBO TX. Es handelt sich dabei um eine grafische Programmiersprache, in der der Programmfluss durch aneinandersetzen von Programmbausteinen zu einem Programmablaufplan strukturiert ist.

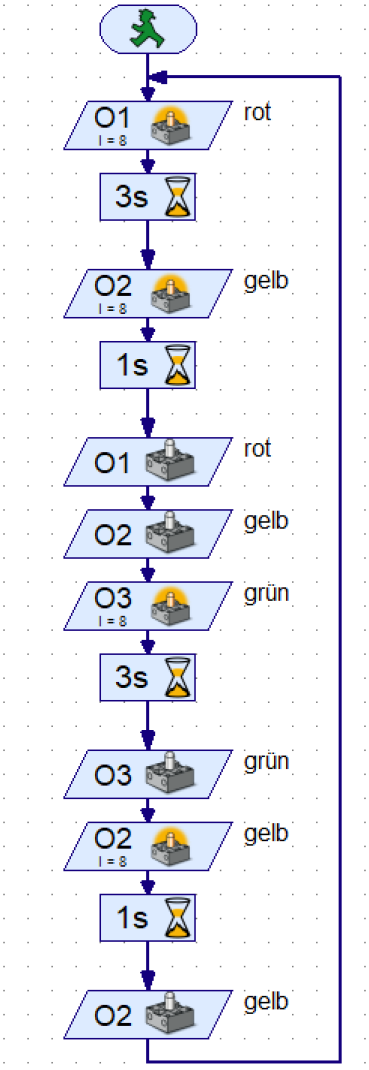


Abbildung 3.1: Ampelsteuerung in ROBOPro

Alle notwendigen Funktionalitäten, wie Verzweigungen, Schleifen, arithmetische Operationen und Ansteuerung der Ein- und Ausgänge sind in ROBOPro in vorgefertigten Bausteinen enthalten, bzw. werden über die Pfeilverbindungen zwischen den Bausteinen realisiert.

In Abbildung 3.1 ist ein beispielhafter Ablauf einer Ampelsteuerung abgebildet. Kleine ROBOPro Programme sind auch für Programmiereinsteiger verständlich. Bei größeren Programmen kann auch bei erfahreneren Entwicklern die Übersichtlichkeit über des Programmes verloren gehen, da aus dem Programm nicht immer erkennbar ist, welches Bauteil an welchem Pin des TXT-Controller angeschlossen ist.

Die bei ROBOPro mitgelieferte Standardbibliothek besitzt nicht den Umfang, wie dies bei herkömmlichen Programmiersprachen der Fall ist. Aus diesem Grund kann es vorkommen, dass bei komplexeren Projekten die Programmierung in ROBOPro nicht möglich oder mit größerem Aufwand verbunden ist. ROBOPro unterstützt sowohl den Onlinebetrieb als auch den Downloadbetrieb der angeschlossenen Modelle.

3.3 Scratch

Scratch ist eine grafische Programmiersprache, die für Programmiereinsteiger entwickelt wurde. Die Ursprüngliche Zielgruppe ist jedoch nicht das Entwickeln von Steuerungen für den TXT-Controller, sondern das Bewegen einer Katze. Der Programmfluss wird in Scratch mithilfe von Struktogrammen gesteuert. In Scratch ist sowohl der Onlinebetrieb als auch der Downloadbetrieb von fischertechnik-Modelle möglich.

Die Entwicklung eines Programmes für den TXT-Controller mit Scratch läuft im Browser ab. Die Website ftscratch3[2] stellt eine grafische Entwicklungsoberfläche zur Verfügung, mit der direkt Scratchprogramme entwickelt werden können.

Um Programme für fischertechnik-Modelle mit einem TXT-Controller in Scratch entwickeln zu können, muss zuerst die fischertechnik TXT Erweiterung eingebunden werden. Dabei werden die benötigten Bausteine für die Ansteuerung von Motoren und Sensoren importiert. Die Übersichtseite möglicher Erweiterungen wird mit dem Klick auf die Schaltfläche *Erweiterung hinzufügen* unten links geöffnet. Anschließend kann die Scratch-Erweiterung für den TXT-Controller eingebunden werden.

Nachdem ein Programm entwickelt wurde, kann dieses auf den Rechner heruntergeladen werden und mit dem Webinterface des TXT-Controller auf diesen hochgeladen werden. Nach erfolgtem Upload kann das Programm über die Toucheingabe des TXT geladen und gestartet werden.

Um ein Scratchprogramm im Onlinebetrieb zu starten wird die Software FTScratchTXT[3] benötigt. Dieses Programm übernimmt die Kommunikation zwischen Browser und TXT-Controller. Beim Start des Scratchprogrammes im Browser wird über diese Applikation das Modell gesteuert.

[2] `https://www.ftscratch3.com/`
[3] Download: `https://ftscratch.github.io/ROBO-TXT/bin/FTScratchTXT.exe`

Für Programmieranfänger ist Scratch eine sehr einfache und intuitiv bedienbare Möglichkeit fischertechnik-Modelle mit einem TXT-Controller zu steuern. Es werden keine Programmierkenntnisse benötigt, somit kann man sich beim Entwickeln auf die Funktionalität fokussieren und muss nicht auf korrekte Syntax und Semantik im Quellcode achten. In Abbildung 3.2 ist die Ampelstuerung als Scratchprogramm dargestellt.

Abbildung 3.2: Ampelsteuerung in Scratch

Bei der Programmierung größerer und komplexerer Modelle kommt man mit Scratch jedoch relativ schnell an die Grenzen der Sprache. Dies liegt unter anderem daran, dass mit Scratch nur Programme für Modelle mit nur einem TXT-Controller entwickelt werden können.

3.4 C/C++

Als offiziell Unterstützte Hochsprache von fischertechnik für die Programmierung des TXT-Controllers dient die Sprache C/C++. Mit der für diese Arbeit verwendeten Firmwareversion 4.6.6 für den TXT-Controller ist es nur möglich C/C++ Programme für den Downloadbetrieb zu entwickeln. Für ältere Firmwareversionen gibt es auch die Möglichkeit Programme für den Onlinebetrieb zu schreiben.

Ein sehr großer Vorteil der Sprache C/C++ gegenüber ROBOPro, ist die umfangreiche Standardbibliothek und die vielen weiteren Bibliotheken von Drittanbietern, die die Entwicklung an vielen Stellen vereinfachen kann.

Mit den Header Dateien *FtShmem.h* und *KeLibTxtDl.h* stellt fischertechnik eine Programmierschnittstelle zur Verfügung, mit der auf die Pins des TXT-Controllers zugegriffen werden kann. Jedes Programm für den TXT-Controller, welches auf die IO-Pins zugreift, muss diese einbinden. Um ein Programm für den Downloadbetrieb zu entwickeln, muss zuerst die Funktion *StartTxtDownloadProg()* aufgerufen werden. Diese kann unter Umständen einen Errorcode zurückgeben. Ist dies erfolgreich, kann mit dem Aufruf von *GetKeLibTransferAreaMainAddress()* ein Zeiger auf die *TransferArea* Struktur geholt werden. Die *TransferArea* Struktur ist in der Header Datei *FtShmem.h* definiert und ist für die Ansteuerung der Pins verantwortlich.

Im nachfolgenden Beispielprogramm ist die bereits bekannte Ampelsteuerung in C/C++ abgebildet. Der Zeiger auf die TransferArea ist hierbei, wie auch in späteren Beispielen, in der Variable pTArea hinterlegt.

```
1  #include <unistd.h>          // for sleep()
2
3  #include "KeLibTxtDl.h"      // TXT Lib
4  #include "FtShmem.h"         // TXT Transfer Area
5
6  int main(void) {
7    FISH_X1_TRANSFER *pTArea;
8    if (StartTxtDownloadProg() == KELIB_ERROR_NONE) {
9      pTArea = GetKeLibTransferAreaMainAddress();
10     if (pTArea) {
11       while(true){
12         pTArea->ftX1out.duty[0] = 512;       //red
```

```
13          sleep(3);
14          pTArea->ftX1out.duty[1] = 512;       //yellow
15          sleep(1);
16          pTArea->ftX1out.duty[0] = 0;       //red
17          pTArea->ftX1out.duty[1] = 0;       //yellow
18          pTArea->ftX1out.duty[2] = 512;       //green
19          sleep(3);
20          pTArea->ftX1out.duty[2] = 0;       //green
21          pTArea->ftX1out.duty[1] = 512;       //yellow
22          sleep(1);
23          pTArea->ftX1out.duty[1] = 0;       //yellow
24        }
25      }
26      StopTxtDownloadProg();
27    }
28    return 0;
29  }
```

Listing 3.1: Ampelprogramm in C++

Wie an diesem Beispiel sofort zu sehen ist, ist die Programmierung des TXT mit C/C++ nicht so intuitiv wie mit einer der grafischen Programmiersprachen. Dies liegt zum großen Teil daran, dass ROBOPro und Scratch sich um den Verbindungsaufbau zur IO-Platine kümmern, in C/C++ dies hingegen dem Entwickler überlassen wird. Der Verbindungsaufbau lässt das Programm im ersten Moment komplexer aussehen. Die eigentliche Funktionalität des Programmes ist dagegen leichter zu verstehen. Nehmen wir zum besseren Verständnis eines einzelnen Pins die Zeile

```
1     pTArea->ftX1out.duty[0] = 512;
```

auseinander. Die Zeile ist für das Einschalten des roten Ampellichtes verantwortlich. Zuerst wird mit einer Zeigeroperation auf den Outputbereich der *TransferArea* zugegriffen. Der Outputbereich beinhaltet neben einigen weiteren Attributen (Übersicht im Unterabschnitt 3.4.2) auch das Attribut *duty*, über welches das Tastverhältnis des PWM Ausganges des im Index angegebenen Pins gesetzt werden kann. Hierbei sind ganzzahlige Werte im Intervall von 0 bis 512 erlaubt. Somit können im Vergleich zu ROBOPro und Scratch genauere Werte angegeben werden (PWM Intervall reicht hier von 0 - 8).

3.4.1 Compilieren des Quellcodes

Um eine gute Plattformunabhängigkeit bei der Entwicklung gewährleisten zu können, sollte der Compiler für die ARM Plattform des TXT-Controller in Verbindung mit *Make* genutzt werden. Ist das hierfür benötigte *Makefile* einmal eingerichtet, ist die Entwicklung von Software für den TXT-Controller sehr einfach. Für die Programmierung wird noch ein beliebiger Texteditor benötigt. Um den Compiliervorgang zu vereinfachen, bietet sich ein Editor mit einem integrierten Terminal an, wie z.B. Visual Studio Code. Alle bisher genannten Software-Tools sind sowohl für macOS, Linux und Windows verfügbar. Alternativ können je nach persönlichen Präferenzen auch andere Editoren bzw. Entwicklungsumgebungen genutzt werden.

Um unter Windows Programme für den TXT-Controller kompilieren zu können, wird ein Compiler[4] benötigt, der für die Zielplattform des TXT-Controller geeignet ist. Um bei der Kompilierung lange Compiler-Pfade zu vermeiden, empfiehlt es sich das binary-Verzeichnis des Compilers in die Path-Umgebungsvariable bei Windows einzufügen.

Für Linuxumgebungen, die auf Debian basieren, können Compiler und Make einfach mit folgenden Befehlen über die Kommandozeile installiert werden:

```
1 apt install make
2 apt install g++-arm-linux-gnueabihf
```

Wird innerhalb des Makefiles für den Compileraufruf die Variable CXX (C++ Compilername) genutzt, muss diese am Anfang des *makefile* gesetzt werden. Dies wird mit der Zeile `CXX = arm-linux-gnueabihf-g++` erreicht.

Um Programmieranfängern ein Verständnis über die Aufgaben und Funktionsweisen eines Compilers näherzubringen, kann es ebenfalls sinnvoll sein, den Quellcode mit einem Makefile zu übersetzen. Mit einer bereits vorgefertigten Vorlage ist dies auch für Anfänger bedienbar. Würde stattdessen eine vollständige Integrated Development Environment (IDE) zur Entwicklung genutzt, passiert für Einsteiger viel 'Magie' im Hintergrund, außerdem könnten sie von der Masse an Features einer Entwicklungsumgebung abgeschreckt werden.

[4] `https://releases.linaro.org/components/toolchain/binaries/latest-7/arm-linux-gnueabihf/gcc-linaro-7.5.0-2019.12-i686-mingw32_arm-linux-gnueabihf.tar.xz`

3.4.2 Pinansteuerung über die TransferArea

Die TransferArea aus der Header Datei *FtShmem.h* ist eine Struktur des Datentyps `FISH_X1_TRANSFER` und beinhaltet ihrerseits einige weitere Strukturen als Attribute, die für spezifischere Teilbereiche verantwortlich sind. Mit den vier wichtigsten Attributen können die meisten fischertechnik-Modelle vollständig programmiert werden.

- *ftX1config* und *ftx1state* dienen zur Konfiguration von Pins
- *ftX1in* zur Ansteuerung der Input Pins
- *ftX1out* zur Ansteuerung der Output Pins

In den folgenden Abschnitten wird der Aufbau und die Nutzung der TransferArea näher erläutert. Einige seltene Aspekte sind dabei außen vor gelassen[5].

ftX1config

Bevor ein Universaleingang des TXT-Controller genutzt werden kann, muss dieser konfiguriert werden.

Code	Beschreibung
`pTArea->ftX1config.uni[i].mode = m`	Modus Konfiguration des Universaleinganges an Pin i
`pTArea->ftX1config.uni[i].digital = 1`	Digital-/Analogkonfiguration des Universaleinganges an Pin i (0/1)
`pTArea->ftX1state.config_id++`	Konfiguration übernehmen

Tabelle 3.1: Konfiguration eines Universaleinganges

Um einen Universaleingang zu konfigurieren, ist die Angabe eines Betriebsmodus notwendig:

- Spannung messen: *MODE_U*
- Widerstand messen: *MODE_R*
- Abstand messen: *MODE_ULTRASONIC*

[5] Für interessierte sei an dieser Stelle auf die originale Header Datei FtShmem.h `https://github.com/fischertechnik/txt_demo_c_download/blob/master/deps/include/FtShmem.h` verwiesen.

Neben der Angabe eines Modus ist auch die Art der Signalübertragung für die Konfiguration eiens Universaleinganges notwendig. Es sind folgende Modus / Signalübertragungsart Kombinationen möglich:

Modus	**Signalübertragungsart**	**Sensoren**
MODE_R	digital	Taster, Fototransistor, Reed-Kontakt, ...
MODE_R	analog	Fotowiderstand, NTC...
MODE_U	digital	Spurensensor
MODE_U	analog	Farbsensor, Spannungsmessung
MODE_ULTRASONIC	digital/analog	Ultraschall Abstandssensor

Tabelle 3.2: Konfigurationsmodi der Universaleingänge

Aus dieser Tabelle lässt sich beispielsweise die Konfiguration eines Tasters an Pin I1 ablesen:

```
1 pTArea->ftX1config.uni[0].mode = MODE_R; //  Widerstand
2 pTArea->ftX1config.uni[0].digital = 1;   //  digitales Signal
3 pTArea->ftX1state.config_id++;           //  Konfiguration speichern
```

Eine Konfiguration der Eingänge bei der Programmierung mit ROBOPro und Scratch ist nicht notwendig. Dies wird automatisch durch die jeweiligen Programmiersprachen geregelt.

ftX1in

Mit der Struktur *ftx1in* können Eingangssignale am TXT-Controller ausgelesen werden.

Code	Beschreibung	Datentyp
`pTArea->ftX1in.uni[i]`	Auslesen des Wertes des Universaleinganges (I1-I8) an Pin i	signed short
`pTArea->ftX1in.cnt_in[i]`	Auslesen des Wertes am Zähleingang (C1-C4) an Pin i	bool
`pTArea->ftX1in.counter[i]`	Auslesen des Schrittmotor Counters (C1-C4) an Pin i	signed short
`pTArea->ftX1in.motor_ex_reached[i]`	Hat Schrittmotor an Pin i sein Ziel erreicht?	bool

Tabelle 3.3: Eingangssignale am TXT-Controller lesen

Mit den Informationen dieser Tabelle können Ausdrücke für Bedingungen im Programmcode formuliert werden, hier wird Beispielsweise der Schleifenrumpf ausgeführt, solange am Eingang I1 ein High-Signal[6] anliegt:

```
1 while (pTArea->ftX1in.uni[0]){
2     sleep(10);
3 }
```

Bei der Verwendung von Schleifen zur Abfrage von Sensorwerten sollte im Schleifenrumpf eine kurze Wartezeit überbrückt werden. Ohne diese kommt es im Programmablauf manchmal zu Problemen bei der Ausführung. Eine Wartezeit von 10ms hat sich als ausreichend gezeigt. Der Grund hierfür ist unbekannt.

[6] Beispiel Taster: Schließer und Taster gedrückt, Öffner und Taster nicht gedrückt

ftX1out

Mit den Attributen der Struktur *ftx1out* können die Ausgänge des TXT-Controllers konfiguriert und gesteuert werden.

Code	Beschreibung	Wertebereich
`pTArea->ftX1out.duty[i]=x`	PWM Wert x für den Pin i setzen	$0 \leq x \leq 512$
`pTArea->ftX1out.distance[i]=x`	Distanz x für Schrittmotor an Pin i setzen	$0 \leq x \leq 65535$
`pTArea->ftX1out.master[i]=x`	Synchronisiere Schrittmotor an Pin i mit Schrittmotor an Pin x	$0 \leq x \leq 4$
`pTArea->ftX1out.cnt_reset_cmd_id[i]++`	Counter von Schrittmotor an Pin i zurücksetzen	
`pTArea->ftX1out.motor_ex_cmd_id[i]++`	Motoreinstellungen aktualisieren	

Tabelle 3.4: Ausgangssignale am TXT-Controller schreiben

Soll nun statt einer einfachen Lampe ein Motor angesteuert werden können, werden hierfür zwei Pins benötigt. Jeder Motoranschluss wird mit einem Pin des TXT-Controllers verbunden (z.B. 0 und 1). Wird nun auf Pin 0 ein High-Signal gegeben und gleichzeitig auf Pin 1 ein Low-Signal, dreht sich der Motor. Ist eine Änderung der Drehrichtung erwünscht, muss das High-Signal von Pin 0 in ein Low-Signal und das Low-Signal von Pin 1 in ein High-Signal geändert werden. Dieser Anwendungsfall ist nachfolgend in einem Beispielprogramm abgebildet. Zuerst dreht sich der Motor mit halber Geschwindigkeit in die eine Richtung. Nach einer Sekunde wird die Drehrichtung geändert.

```
1 pTArea->ftX1out.duty[0]=256;
2 pTArea->ftX1out.duty[1]=0;
3 sleep(1000);
4 pTArea->ftX1out.duty[0]=0;
5 pTArea->ftX1out.duty[1]=256;
```

Wie aus den vorigen Abschnitten leicht zu erkennen, ist die Entwicklung von Programmen für den TXT mit der von fischertechnik bereitgestellten Programmierschnittstelle möglich, jedoch mit größerem Aufwand bei der Einarbeitung und Programmierung. Der Entwickler muss immer genau wissen, wie welcher Pin adressiert (und evtl. konfiguriert) werden muss.

3.5 Python

Der TXT-Controller kann auch mit der Sprache Python programmiert werden, dafür wird das Modul *ftrobopy*[7] benötigt, mit welchem man auf die Ein- und Ausgänge des TXT zugreifen kann.

Bei der Entwicklung mit Python ist sowohl Online- als auch Downloadbetrieb des Modells mit identischem Quellcode möglich.

Wenn man die Communityediton der TXT-Firmware[8] verwendet, können mit dem UI Framework PyQt Applikationen für den TXT entwickelt werden, die eine grafische Benutzerschnittstelle besitzen. Somit können Modelle auch über Touch bedienbar gemacht werden.

Die Programmierung des TXT-Controller mit ftrobopy ist noch[9] nicht für den parallelen Betrieb mehrerer TXT-Controller ausgelegt.

Die Ansteuerung von Ein- und Ausgängen mit ftrobopy wird über Objekte gelöst. Durch geschickte Benennung dieser Objekte ist es möglich deutlich lesbareren Code als in C/C++ oder auch ROBOPro und Scratch zu entwickeln. Im beispielhaften Ampelprogramm im Listing 3.2 muss sich der Entwickler nicht merken, welche Ampelfarbe hinter welchem Pin steckt. Um eine bestimmte Lampe einzuschalten, kann er jetzt auf ein Objekt mit klarem Namen zurückgreifen. Ein weiterer großer Vorteil in der Nutzung von Objekten liegt in der Änderbarkeit des entwickelten Programmcodes. Wird aufgrund von Umbaumaßnahmen am Modell eine Änderung der Pinbelegung am TXT durchgeführt, muss der zu adressierende Pin lediglich beim Erzeugen des Objektes angepasst werden. In den anderen Programmiersprachen lässt sich dies teilweise mit der Nutzung von Indexvariablen umsetzen, was aber auch nicht unbedingt zu lesbarem Quellcode führt.

[7] `https://github.com/ftrobopy/ftrobopy`

[8] `https://github.com/ftCommunity/ftcommunity-TXT`, zur Installation der Communityfirmware wird eine Micro SD Karte benötigt

[9] Stand Dezember 2019, Planungen für deren Entwicklung gab es zu diesem Zeitpunkt bereits. Im Februar 2020 wurde mit den Umsetzungen begonnen

```
import sys
import ftrobopy
import time

ftrob = ftrobopy.ftrobopy('auto')
red = ftrob.output(1)
yellow = ftrob.output(2)
green = ftrob.output(3)
while True:
    red.setLevel(512)
    time.sleep(3)
    yellow.setLevel(512)
    time.sleep(1)
    red.setLevel(0)
    yellow.setLevel(0)
    green.setLevel(512)
    time.sleep(3)
    green.setLevel(0)
    yellow.setLevel(512)
    time.sleep(1)
    yellow.setLevel(0)
```

Listing 3.2: Ampelprogramm in Python

Die Konfiguration von Eingängen ist in diesem Beispiel nicht zu sehen, wird aber ebenfalls von ftrobopy übernommen und ist somit genauso einfach wie in den grafischen Programmiersprachen.

3.6 Vergleich der untersuchten Sprachen

Um zu einem endgültigen Resultat zu kommen, welche der untersuchten Sprachen für die Entwicklung einer Steuerung für die Modellfabrik am besten geeignet ist, werden einige Kriterien gegenübergestellt. Die wichtigen Merkmale sind die Ansteuerung eines Extension-TXT-Controller vom Master-TXT-Controller, sowie eine umfangreiche Standardbibliothek bzw. Drittanbieterbibliotheken.

Weitere berücksichtigte Merkmale betreffen die Einarbeitung bzw. Anfängerfreundlichkeit der Sprache. Diese stellen, im Gegensatz zu den bereits genannten, keine KO-Kriterien dar.

Funktion	**ROBOPro**	**Scratch**	**C/C++**	**Python**
Unterstützung mehrerer TXT	ja	nein	ja	nein
Umfangreiche Bibliotheksunterstützung	nein	nein	ja	ja
Für Anfänger geeignet	ja	ja	nein	ja
Schnelle Einarbeitung möglich	ja	ja	nein	ja
Relevanz für andere Vorlesungen	nein	nein	ja	ja
sowohl Online- als auch Downloadbetrieb	ja	ja	nein	ja
Konfiguration von Eingängen notwendig	nein	nein	ja	nein
Benennung von Ein-/Ausgängen mit Objekten	nein	nein	nein	ja
Änderbarkeit von Programmcode	aufwändig	aufwändig	aufwändig	neutral
Lesbarkeit von Programmcode	neutral	gut	schlecht	gut

Tabelle 3.5: Vergleich der Programmiersprachen

Wenn man anhand dieser Gegenüberstellung der einzelnen Programmiersprachen eine Entscheidung zur weiterverwendeten Programmiersprache treffen muss, sticht Python im ersten Moment als klarer Sieger heraus, da Python sehr viele Kriterien erfüllt. Da für die Steuerung der Modellfabrik an zwei der drei Einheiten am Master TXT-Controller ein weiterer Extension TXT-Controller angeschlossen ist, muss die Programmiersprache

zwingend die Entwicklung von Programmen für mehrere TXT-Controller unterstützen. Mit dieser Anforderung sind Python und Scratch ungeachtet ihrer verbleibenden Vorteile bereits vorzeitig aus dem Rennen geschieden [10].

Um die Modellfabrik zukunftsfähig im Industrie 4.0 Bereich weiterentwickeln zu können, wird man früher oder später auf weitere Technologien angewiesen sein. Diese müssen auf einfachem Weg in Form von Bibliotheken und Frameworks in den bereits existierenden Programmcode integriert werden können. Da die darüber hinaus untersuchten Merkmale im Gegensatz zu den bisher betrachteten Kriterien nicht als KO-Kriterien gewertet werden, kann bereits anhand der Ersten beiden untersuchten Merkmale eine Entscheidung getroffen werden.

Im Punkt *umfangreiche Bibliotheksunterstützung* ist C/C++ ROBOPro deutlich überlegen. Dadurch steht fest, dass die Steuerung der Modellfabrik in der Sprache C/C++ umgesetzt wird.

[10] Unter Umständen kann Python in Zukunft trotzdem für die Programmierung der Modellfabrik berücksichtigt werden, da während der Entstehungszeit dieser Studienarbeit mit der Erweiterung von ftrobopy um die Multi-TXT-Controller Unterstützung begonnen wurde. Dies ist in der Bewertung berücksichtigt.

4 Entwicklung der Programmierschnittstelle

Aus dem Programmiersprachenvergleich im vorigen Kapitel geht C/C++ als Sieger hervor. Da die Entwicklung von Programmen in C/C++ für den TXT-Controller mit längerer Einarbeitungszeit in die bestehende Programmierschnittstelle und weiteren Schwierigkeiten behaftet ist, werden in diesem Kapitel weitere Abstraktionsebenen entworfen, die diese Probleme beheben. Nachtiele, die bei der Programmierung des TXT-Controller in C/C++ gegenüber anderen Programmiersprachen bestehen und durch diese Abstraktion behoben werden sollen, sind:

- Konfiguration von Eingängen notwendig
- Hoher Aufwand bei Änderung und Wartung
- Hohe Einarbeitungszeit
- Schlecht lesbarer Code

Neben diesen, für Programmiereinsteiger ohnehin schon großen Hürden, werden bei der Verwendung der bestehenden Programmierschnittstelle teilweise Kenntnisse in Zeigerarithmetik benötigt, um Pins am TXT-Controller ansteuern zu können. Das benötigte Wissen in diesem Bereich ist nicht tief, wird jedoch beim Lernen von bewährten Einsteigersprachen wie Java, Python oder C# nicht vermittelt, da die Sprachen dies nicht direkt unterstützen.

Die Hauptziele der Einführung der weiteren Abstraktionsebenen sind einerseits eine leichtere Einarbeitung in die Programmierung des TXT-Controller und andererseits die Verbesserung der Lesbarkeit des Quellcodes, sowie eine bessere Wartbarkeit des entwickelten Programmcodes.

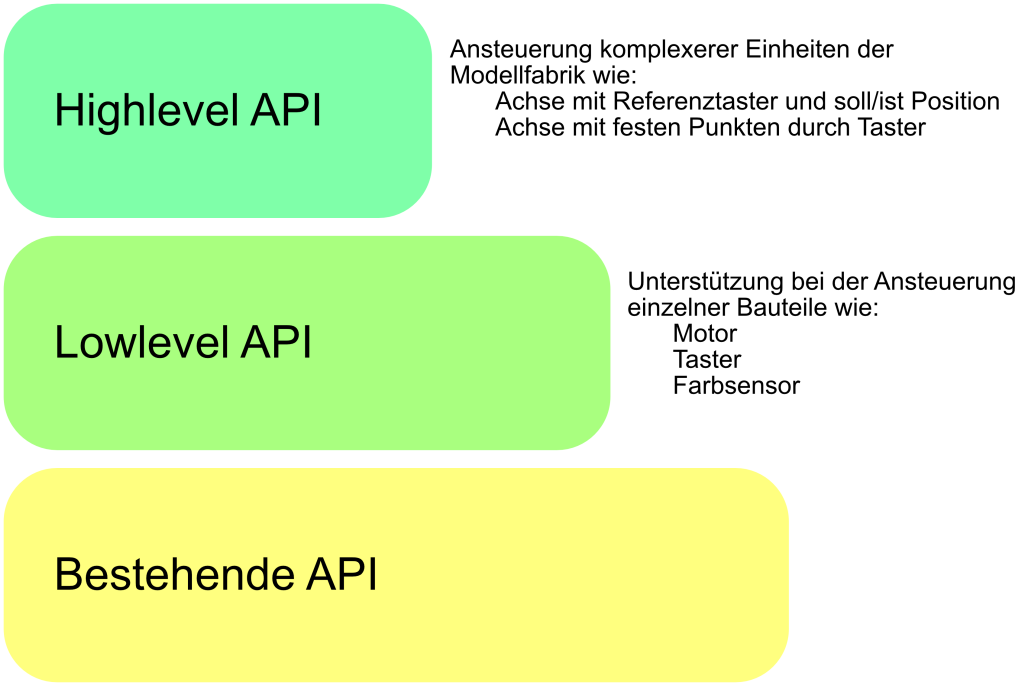


Abbildung 4.1: Abstraktionsebenen der Programmierschnittstelle

Aufbauend auf der bereits von fischertechnik bereitgestellten Schnittstelle wird zuerst eine Lowlevel API entworfen. Diese ist verantwortlich für die Ansteuerung einzelner elektronischer Bauteile wie Motoren, Lampen und Sensoren. Die Lowlevel API stellt somit ein Fundament dar, auf dem beliebige größere Programme für den TXT-Controller entwickelt werden können.

Als Grundlage für die Highlevel API dient die Lowlevel API. In der Highlevel API werden komplexere Anwendungsfälle abstrahiert, die nicht wie die Lowlevel API auf allgemeine Programme ausgelegt sind, sondern für spezielle Use-Cases der Modellfabrik gedacht sind.

Zusätzlich zu Lowlevel- und Highlevel API gibt es noch die Utils API, welche einige weitere nützliche Funktionen bereitstellt. In den folgenden Abschnitten werden die einzelnen Schnittstellen und deren Verwendung detailliert erläutert.

4.1 TXT Lowlevel API

Die Lowlevel API für den TXT-Controller baut auf der von fischertechnik bereitgestellten Programmierschnittstelle auf und ermöglicht die Steuerung einzelner Aktoren und Sensoren.

Die konkreten Anforderungen an die Lowlevel API sind:

- Verringerung von Overhead im Quellcode
- Automatische Konfiguration der Sensoreneingänge
- Wiederverwendbare Objekte für die Ansteuerung gleicher Pins
- Verbesserung der Wartbarkeit
- Verständliche Namen der enthaltenen Klassen, Attribute und Methoden

4.1.1 Overheadverringerung und die Klasse TXT

Jedes C/C++ Programm für den TXT-Controller benötigt zu Beginn einen Funktionsaufruf, welcher das Programm im Downloadbetrieb startet und einen, der die Speicheradresse der TransferArea der IO-Platine ermittelt. Nachdem der eigentliche Programmablauf beendet ist, wird durch einen weiteren Funktionsaufruf der Downloadbetrieb wieder beendet. Die hierfür benötigten Funktionsaufrufe, Überprüfungen, etc. fügen jedem Programm zusätzlichen Overhead hinzu, was zu schwerer lesbarem Quellcode führt. Im C/C++ Ampelbeispiel Listing 3.1 wird dabei ein Overhead von 9 Zeilen erzeugt, wobei die eigentliche Funktionsweise der Ampelsteuerung nur 14 Zeilen in Anspruch nimmt. Dieser Overhead an Überprüfungen und Aufrufen wird in jedem Programm in identischer Form benötigt, was einerseits einen erhöhten Aufwand bedeutet und andererseits das Programm unübersichtlicher macht.

Um dies zu vereinfachen wird die Klasse `TXT` entwickelt. In deren Konstruktor und Destruktor werden die initialen Funktionsaufrufe und Überprüfungen ausgeführt. Somit wird für ein Programm mit gleicher Funktionsweise nur noch eine Zeile zum erzeugen des `TXT` Objektes in der `main()` Methode benötigt:

```
#include "TXT_lowlevel_API.h"
main(){
    TXT txt = TXT();
    //gewuenschter Programmablauf
}
```

Listing 4.1: Programm mit geringerem Overhead

Der TXT-Controller besitzt einen eingebauten Lautsprecher. Über den Aufruf der Methode `txt.playSound(1, 2)` können die auf dem TXT hinterlegten Geräusche abgespielt werden. In diesem Fall wird das Geräusch mit dem Index 1 zweimal nacheinander abgespielt. Insgesamt hat der TXT Standardmäßig 29 Geräusche.

Da die Lowlevel API nicht alle Funktionalitäten der von fischertechnik bereitgestellten Schnittstelle abstrahiert, kann es sein, dass für einen selteneren Anwendungsfall keine Implementierung in der Lowlevel API enthalten ist. Mit dem Aufruf der Methode `txt.getTransferArea()` kann auf die TransferArea zugegriffen werden und dadurch trotzdem eine Lösung für dieses speziellere Problem entwickelt werden.

4.1.2 Konfiguration der Eingänge I1-I8 / C1-C4

Um beim Entwickeln von Programmen für den TXT-Controller keine manuelle Konfiguration der angeschlossenen Eingänge vornehmen zu müssen, bietet die Lowlevel API eine Reihe von Klassen an, deren Objekte die Konfiguration und Verwendung der jeweiligen Sensoren unterstützen. Die Konfiguration der Sensorobjekte wird im jeweiligen Konstruktor anhand der in Tabelle 3.2 aufgezeigten Konfigurationsmöglichkeiten vorgenommen. Dadurch entfällt der manuelle Konfigurationsschritt der Universaleingänge.

Für die Erzeugung eines Sensorobjektes (wie auch für die Erzeugung von Ausgangsobjekten) existieren zwei Alternativen. Zum einen kann ein Eingangsobjekt durch den Aufruf des Konstruktors erzeugt werden. Dabei werden als Übergabeparameter die TransferArea des TXT-Controllers und die Pinnummer benötigt. Auf die TransferArea kann man über das `TXT`-Objekt mit dem Aufruf von `txt.getTransferArea()` zugreifen. Die kürzere Alternative kapselt den Konstruktoraufruf des Eingangobjektes in einer Methode des `TXT` Objektes. Bei diesem Methodenaufruf wird nur die Pinnummer benötigt und der Konstruktoraufruf wird in der Fabrikmethode um die TransferArea als Parameter ergänzt.

```
1 TXT txt = TXT();
2 DigitalInput taster1 = DigitalInput(txt.getTransferArea(),1);
3 //alternativ
4 DigitalInput taster2 = txt.digitalInput(1);
5 if(taster1.value()){
6     //Wenn Taster 1 gedrueckt (Taster muss als Schliesser angeschlossen
    sein)
7 }
```

Listing 4.2: Ansteuerung von Eingängen

Insgesamt existieren sieben unterschiedliche Klassen für die Ansteuerung der Universaleingänge I1-I8 und eine weitere Klasse für die Ansteuerung der Zähleingänge C1-C4.

Klasse	**Anwendungsfall**	**Sensoren**	**Pins**
DigitalInput	Digitaler Zustand messen	Taster, Fototransistor, ...	I1-I8
AnalogInput	Widerstandsmessung		I1-I8
NTC	Temperaturmessung	NTC	I1-I8
ColorSensor	Fraberkennung	Farbsensor	I1-I8
Voltage	Spannungsmessung		I1-I8
Ultrasonic	Distanzmessung	Ultraschallabstandssensor	I1-I8
TrackSensor	Spurerkennung	Spurensensor	I1-I8
Counter	schnelle Zählintervalle	Taster	C1-C4

Tabelle 4.1: Klassen für die Steuerung der Eingänge

Die Sensorwerte können mit der Methode `value()` der Sensorobjekte abgefragt werden. Bis auf den Spurensensor wird jeder Sensor an einem Universaleingang angeschlossen. Damit der Spurensensor Werte auf der rechten und linken Seite unterscheiden kann, benötigt dieser zwei Universaleingänge. Zum Einlesen dieser Werte gibt es die Methoden `valueLeft()` und `valueRight()`. Beim Aufruf von `taster.waitFor(DigitalState::HIGH)` auf ein *DigitalInput* Objekt wird der Programmablauf so lange angehalten, bis an dem jeweiligen Objekt ein High-Signal anliegt. Die Enumeration *DigitalState* besitzt Werte für *HIGH* und *LOW*. Ein Objekt der Klasse *ColorSensor* besitzt zusätzlich die

Methode `color()`, welche den aktuellen Farbwert in einen Wert der Enumeration *Color* konvertiert und zurückgibt. Mögliche Rückgabewerte sind: *BLUE*, *RED* und *WHITE*. Der Widerstand eines NTC Sensors wird im *NTC* Objekt durch Aufruf der Methode `getTemperature()` in Werte der Celsius-Skala umgerechnet. Für einen angeschlossenen Taster (oder anderer digitaler Sensor) an den Zähleingängen C1-C4 kann mit der Methode `waitSteps(20)` auf ein *Counter* Objekt so lange gewartet werden, bis der angeschlossene Sensor eine bestimmte Anzahl an Impulsen erkannt hat. In diesem Beispiel wird solange gewartet, bis der Taster 10 mal gedrückt und 10 mal losgelassen wird, da sowohl positive als auch negative Flanken beachtet werden.

Die Datentypen der Rückgabewerte und weitere Methoden der Sensoren, sowie die Fabrikmethoden im *TXT* Objekt zur Bildung der Sensorobjekte lassen sich aus dem Klassendiagramm in Abbildung 4.2 der Lowlevel API entnehmen.

4.1.3 Ansteuerung der Ausgänge O1-O8 / M1-M4

Für die Ansteuerung der Ausgänge existieren drei unterschiedliche Klassen. Dabei wird zwischen der Ansteuerung der Ausgänge O1-O8 und der Motorausgänge M1-M4 unterschieden. Für Motorsteuerungen existieren zwei Implementierungen: Normale Motoren, die links und rechts drehen können und Encodermotoren (Schrittmotor), die eine bestimmte Distanz/Schrittanzahl zurücklegen können. Zwei Encodermotoren können miteinander synchronisiert werden, damit diese mit identischer Geschwindigkeit die gleiche Schrittanzahl zurücklegen. Im folgenden Beispielprogramm werden die beiden Schrittmotoren *em1* und *em2* miteinander synchronisiert, und bewegen sich anschließend für 400 Schritte mit der vollen Geschwindigkeit. Eine Umdrehung eines Motors entsprechen 75 Schritte. Mit der Methode `waitToEnd()` wird so lange gewartet, bis der Schrittmotor sein Ziel erreicht hat.

```
1 TXT txt = TXT();
2 EncoderMotor em1 = txt.encoderMotor(1);
3 EncoderMotor em2 = txt.encoderMotor(2);
4 em1.synchronizeTo(em2);
5 em1.distanceLeft(400, 512);
6 em2.left(512);
```

```
em1.waitToEnd();
```

Listing 4.3: Synchronisation von Encodermotoren

Datentypen von Parametern und weitere Methoden von *Motor*, *EncoderMotor* und *Output* sind ebenfalls dem Klassendiagramm in Abbildung 4.2 zu entnehmen.

4.1.4 Verwendung eines Extension Controllers

Bei der Steurung größerer fischertechnik-Modelle können unter Umständen die Anschlüsse eines einzelnen TXT-Controllers nicht ausreichen. Ein Beispiel hierfür ist die in dieser Arbeit behandelte Modellfabrik. Zwei der drei Einheiten der Modellfabrik werden mit jeweils zwei TXT-Controllern gesteuert. Dabei nimmt einer der TXT-Controller die Rolle *Master*, der andere die Rolle *Extension* an. Dies kann am TXT-Controller unter *Settings -> Role* eingestellt werden. Master und Extension werden mit einem Kabel[1] über den *EXT* Anschluss am TXT-Controller verbunden. Programmiertechnisch gibt es zwei Möglichkeiten Anschlüsse des Extension TXT-Controllers anzusteuern:

```
TXT txt = TXT();
//alternative 1
TXT extension = txt.extension();
Output lampe = extension.output(1);
//alternative 2
Output lampe2 = txt.output(10);
lampe.on();
lampe2.on();
```

Listing 4.4: Zugriff auf Extension-TXT

Einerseits kann vom *TXT* Objekt des Masters ein *TXT* Objekt der Extension erzeugt werden, indem die Methode `extension()` aufgerufen wird. Von diesem Extension *TXT* Objekt können genau wie vom normalen *TXT* Objekt Eingangs und Ausgangsobjekte erstellt werden. Die zweite Möglichkeit ist der Aufruf der Fabrikmethoden auf das Objekt des *Master-TXT*, mit höheren Pinnummern. Die Pinnummern I1-I8 am Extension Controller entsprechen den Pinnummern 9-16, analog dazu die Ausgänge O1-O8. Mo-

[1] Das hierfür benötigte 10-Polige Verbindungskabel liegt bei jedem TXT-Controller bei

torausgänge M1-M4 sowie die Zähleingänge C1-C4 am Extension Controller werden über die Pinnummern 5-8 am Master TXT adressiert.

4.1.5 Aufbau und Struktur der Lowlevel API

Die Lowlevel API besteht insgesamt aus 12 Klassen und zwei Enumerationen. Davon sind acht Klassen für die Konfiguration und Ansteuerung der Eingänge verantwortlich und drei Klassen für die Ausgänge. Die verbleibende Klasse *TXT* beinhaltet Fabrikmethoden zur Bildung der Ein- und Ausgangsobjekte, sowie Methoden zu Soundausgabe und Messung der Betriebsspannung des TXT-Controllers. Alle Klassen, welche für eine Ansteuerung eines IO-Pins verantwortlich sind, haben die gemeinsame Oberkalsse *IOPin*. *TrackSensor* ist hiervon ausgenommen, da er über zwei Pins angestuert wird. Aus Übersichtlichkeitsgründen sind die Vererbungspfeile zur Klasse *IOPin* weggelassen.

Im folgenden Klassendiagramm ist die gesamte Klassenstruktur der Lowlevel API abgebildet.

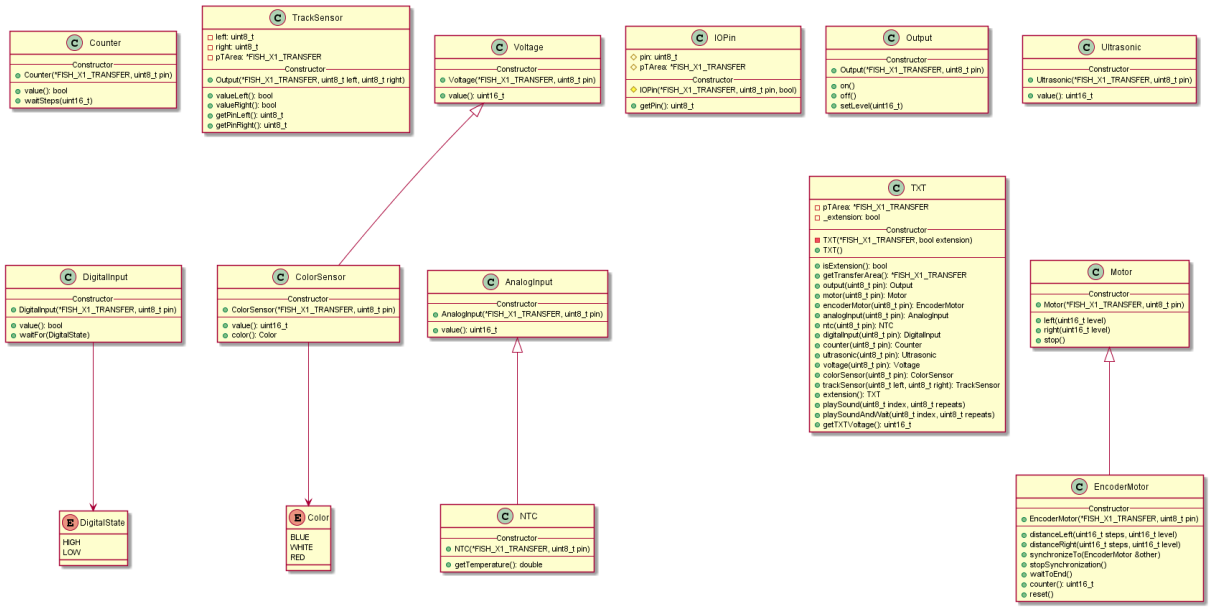


Abbildung 4.2: Klassendiagramm der Lowlevel API

Das Klassendiagramm aus Abbildung 4.2 ist im Anhang in einer größeren Ausführung enthalten.

4.1.6 Vergleich Programmierung vor und nach Lowlevel API

Für einen besseren Vergleich der Programmierung in C/C++ mit und ohne Lowlevel API ist nachfolgend das bereits bekannte Programmierbeispiel der Ampelsteuerung abgebildet.

```
#include "TXT_lowlevel_API.h"

int main(void)
{
    TXT txt = TXT();
    Output red = txt.output(1);
    Output yellow = txt.output(2);
    Output green = txt.output(3);
    while (true)
    {
        red.on();
        sleep(3s);
        yellow.on();
        sleep(1s);
        red.off();
        yellow.off();
        green.on();
        sleep(3s);
        green.off();
        yellow.on();
        sleep(1s);
        yellow.off();
    }
    return 0;
}
```

Listing 4.5: Ampelprogramm in C++ mit Lowlevel API

Im Vergleich zur C/C++ Ampelsteuerung ohne Lowlevel API im Listing 3.1 fällt eine deutliche Verbesserung bei der Lesbarkeit durch verständlichere Methoden und Klassennamen sowie durch Auslagerung von Initialisierungen in die Klasse *TXT* auf.

Die zu Beginn dieses Unterkapitels definierten Ziele werden durch Verwendung der Lowlevel API erreicht. Der Overhead von Initialisierungsaufrufen wird durch die Klasse

TXT verringert. Die Konfiguration der Universaleingänge wird durch die Verwendung von Sensorobjekten automatisch vorgenommen. Die Lesbarkeit des Programmcodes erhöht sich durch klar benannte Klassen und deren Methodenaufrufe. Die Verwendung von Objekten zur Ansteuerung der Ein- und Ausgänge erhöht die Lesbarkeit des Programmcodes bei Vergabe von guten Objektnamen ebenfalls und verbessert gleichzeitig die Wartbarkeit des produzierten Quellcodes.

4.2 TXT-Highlevel API

Die im vorigen Unterkapitel vorgestellte Lowlevel API vereinfacht die Entwicklung von Programmen für den TXT-Controller. Kleinere und sehr allgemeine Programme lassen sich mit dieser leicht umsetzen. Bei einem größeren Programm wie der Modellfabrik treten jedoch einige Anwendungsfälle mehrfach auf. Ist nur die Lowlevel API vorhanden müssen diese mehrfach Implementiert werden. Für größere Modelle ist es deshalb sinnvoll diese häufiger auftretenden Anwendungsfälle in einer weiteren Abstraktionsschicht zu implementieren.

Die hier vorgestellte Highlevel API basiert dabei auf der Lowlevel API und abstrahiert die häufiger auftretenden Anwendungsfälle der Modellfabrik um deren spätere Implementierung zu vereinfachen.

Mehrfach auftretende Anwendungsfälle in der Modellfabrik, die durch die Highlevel API vereinfacht werden sollen sind:

- Steuerung von Achsen mit einem Referenzschalter und Encodermotoren zur Positionsbestimmung (Hochregallager, Vakuumroboter)
- Steuerung von Achsen mit mehreren Tastern, welche jeweils einen Punkt auf der Achse markieren (Hochregallager, Bearbeitungseinheit)

4.2.1 Achse mit einem Referenztaster

Im Hochregallager existieren zwei Achsen mit jeweils einem Referenztaster für die X- und Y-Achse. Im Vakuumroboter befinden sich drei solcher Achsen. Sie haben im allgemeinen folgenden Aufbau:

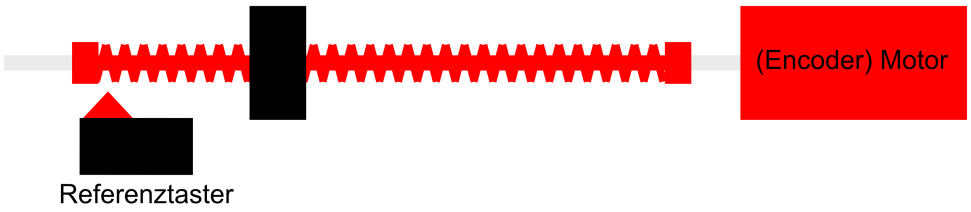


Abbildung 4.3: Achse mit einem Referenztaster

Der Motor bewegt eine Achse, auf der sich eine Schnecke befindet. Durch die Drehbewegung der Achse bewegt sich der in dieser Grafik schwarz abgebildete Baustein. Sobald der Baustein den Referenztaster betätigt, ist die Position 0 erreicht. Von diesem Punkt aus können durch absolute und relative Distanzangaben an die Achse beliebige Positionen angefahren werden. Anstelle eines Encodermotors wäre auch ein normaler Motor denkbar, welcher einem weiteren Taster mit einem Impulszahnrad bei jeder Umdrehung mehrere Impulse gibt. Dieses Szenario tritt in der Modellfabrik an keiner Achse auf, fischertechnik hat jedoch einen anderen Baukasten[2] im Sortiment, bei dem Achsenpositionen mit einem Impulszahnrad bestimmt werden.

Um die Kompatibilität zu diesem Baukasten gewährleisten zu können, existieren in der Highlevel API Klassen, mit der beide Ansätze realisiert werden können: *AxisXS* für Achsen mit normalen Motoren (Motor XS) mit einem Impulszahnrad und *AxisEM* für Achsen mit einem Encodermotor. Die Schnittstelle für die Ansteuerung einer solchen Achse nach außen hin ist für beide Ansätze identisch, da sie das Interface *PosAxis* implementieren.

Eine Achse hat zu jedem Zeitpunkt einen von vier Zuständen: *UNREFERENCED*, *READY*, *LEFT*, *RIGHT*. Mögliche Zustandsübergänge zwischen diesen Zuständen werden im folgenden Diagramm aufgezeigt:

[2]TXT Automation Robots: `https://www.fischertechnik.de/de-de/produkte/spielen/robotics/511933-txt-automation-robots`

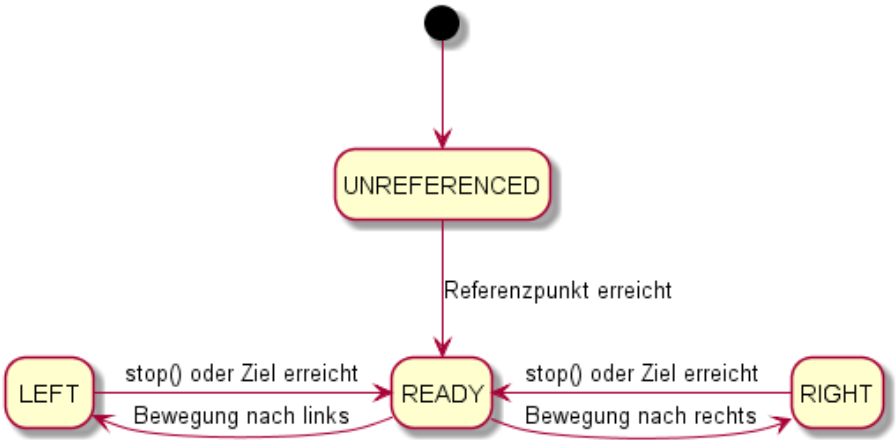


Abbildung 4.4: Zustandübergangsdiagramm einer Achse mit Referenzschalter

Eine Bewegung der Achse nach links wird dabei durch eine negative Anzahl an Schritten, eine Rechtsbewegung durch eine positive Schrittanzahl relativ zur aktuellen Position hervorgerufen. Im folgenden Beispielcode wird eine Folge von Bewegungen einer X-Achse ausgeführt. Die Bewegung einer Achse soll auch asynchron zu weiterem Programmcode ausgeführt werden können, aus diesem Grund kann nach dem Starten der Bewegung mit `moveAbsolutAsync(position)` und `moveRelativeAsync(distanz)` direkt mit weiterem Programmcode fortgefahren werden. Beide Methoden geben ein Objekt von `std::thread` zurück. Bei Bedarf kann zu einem späteren Zeitpunkt mit dem Aufruf der Methode `thread.join()` auf die Beendigung der Achsenbewegung gewartet werden.

```
1  TXT txt = TXT();
2  //Encodermotor an M1, Referenztaster an I2
3  AxisEM xaxis = AxisEM{txt, 1, 2};
4  //state: UNREFERENCED
5  xaxis.reference();
6  //state: READY
7  xaxis.moveAbsolut(600); //state: RIGHT
8  //state: READY
9  std::thread t = xaxis.moveRelativeAsync(-200);
10 //state: left
11 t.join();
12 //state: READY
```

Listing 4.6: Steuerung einer Achse mit einem Referenztaster

Es ist möglich im Konstruktor der Achsen eine maximale Position anzugeben, die die Achse nicht überschreiten darf. Beim Starten einer Bewegung (relativ oder absolut) wird überprüft, ob sich die Zielposition zwischen 0 und der maximal erlaubten Position befindet. Wenn die Zielposition außerhalb des Intervalls $[0, max]$ ist, wird die Bewegung nicht ausgeführt. Wird kein Maximalwert angegeben, wird auf das Intervall $[0, 65536]$ überprüft.

Taster von fischertechnik können sowohl als Öffner als auch als Schließer verwendet werden. Da dies je nach Achse variieren kann, ist es möglich im Konstruktoraufruf einen Modus für den Referenztaster anzugeben. Mögliche Werte sind: *ButtonMode::CLOSER* und *ButtonMode::OPENER*. Standardmäßig wird der Referenztaster als Schließer behandelt.

4.2.2 Achsen mit mehreren Positionstastern

Neben der bereits vorgestellten Achsenart gibt es im Hochregallager und der Bearbeitungseinheit der Modellfabrik eine weitere Achsenart. Im Gegensatz zur bereits vorgestellten Achse mit einem Referenztaster können mit dieser Achse jedoch nicht beliebige Positionen auf der Achse angefahren werden. Die Achsen haben folgenden Aufbau:

Abbildung 4.5: Achse mit mehreren Positionstastern API

Zur Markierung der Zielpunkte der Achse besitzt diese zwei (bzw. allgemeiner N) Taster. Für das Anfahren einer Position wird die Achse so lange bewegt, bis der Taster an der gewünschten Position gedrückt wird. Bei dieser Achsenart ist keine genaue Positionierung zwischen den Positionstastern vorgesehen. Aus diesem Grund wird hierfür auch kein Schrittmotor benötigt.

Der Roboter im Hochregallager sowie der Zustellroboter und der Ofentisch in der Bearbeitungsstation haben jeweils eine solche Achse mit zwei Positionstastern. Der Dreh-

und Bearbeitungstisch in der Bearbeitungsstation wird mit drei Positionstastern positioniert.

Damit für beide Fälle eine möglichst einfache Lösung angeboten werden kann, existiert in der Highlevel API eine Klasse, die Achsen mit einer beliebigen Anzahl an Positionstastern steuern kann: *NRefAxis*. Achsen mit mehreren Referenztastern besitzen ebenfalls zu jedem Zeitpunkt einen Zustand, die Übergänge zwischen den Zuständen ist dabei identisch wie die Übergänge bei einer Achse mit nur einem Referenzschalter in Abbildung 4.4. Die Referenzposition liegt beim zuerst angegebenen Positionstaster. Eine Bewegung nach links wird gestartet, wenn der Index der Zielposition kleiner dem Index der aktuellen Position ist. Wenn der Index größer ist, wird die Achse nach rechts bewegt.

```
1  TXT txt = TXT();
2  //Motor an M1, Positionstaster an I1, I2, I3
3  NRefAxis ax = NRefAxis{txt, 1, std::vector<uint8_t>{1, 2, 3}};
4  //state: UNREFERENCED
5  ax.reference();
6  //state: READY
7  ax.pos(1); //state: RIGHT
8  //state: READY
9  std::thread t = ax.posAsync(0);
10 //state: left
11 t.join();
12 //state: READY
```

Listing 4.7: Steuerung einer Achse mit beliebig vielen Positionstastern

Das Anfahren einer Position wird durch den Aufruf der Methode `pos(index)` gestartet. Als Parameter wird der Taster am gewünschten Index angefahren. In diesem Beispiel wird zuerst referenziert, also der Taster an I1 angefahren. Anschließend wird der Taster an I2 und erneut I1 angefahren.

Die Taster in einer solchen Achse sind standardmäßig ebenfalls auf Schließer eingestellt. Ist ein anderer Modus gewünscht, kann durch den Methodenaufruf `setButtonMode (index,mode)` der Taster an dem jeweiligen Index auf den gewünschten Modus gesetzt werden.

Um die Achsensteuerung für Achsen mit nur zwei Positionstastern zu vereinfachen, existiert eine weitere Klasse, welche von *NRefAxis* abgeleitet ist: *TwoRefAxis*. Diese Unterscheidet sich zu NRefAxis dahingehend, dass nach dem Initialisierungsvorgang der Achse keine Referenzfahrt durchgeführt werden muss. Egal an welcher Position sich der Bewegungskopf der Achse befindet, durch Aufruf von `pos1()` oder `pos2()` kann die gewünschte Zielposition erreicht werden. Somit unterscheidet sich das Zustandsübergangsdiagramm dieser Achse von den anderen.

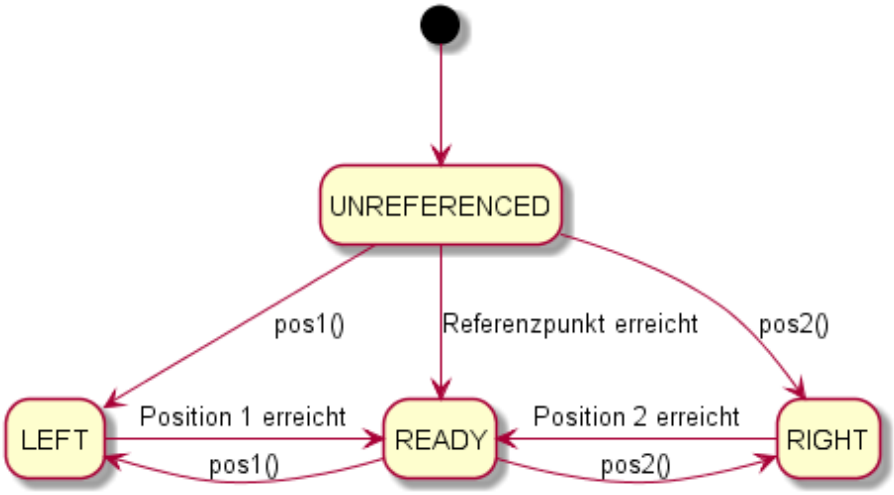


Abbildung 4.6: Zustandübergangsdiagramm mit zwei Positionstastern

Eine Achse mit zwei Referenztastern kann wie im nachfolgenden Beispiel gezeigt gesteuert werden:

```
1  TXT txt = TXT();
2  //Motor an M1, Positionstaster an I2 und I3
3  TwoRefAxis ax = TwoRefAxis{txt, 1, 2, 3};
4  //state: UNREFERENCED
5  ax.pos2(); //state: RIGHT
6  //state: READY
7  std::thread t = ax.pos1Async();
8  //state: left
9  t.join();
10 //state: READY
```

Listing 4.8: Steuerung einer Achse mit zwei Positionstastern

4.2.3 Klassenstruktur der Highlevel API

Die Klassenstruktur der Highlevel API ist in dem folgenden Klassendiagramm dargestellt. Im Anhang ist dieses Klassendiagramm in einer größeren und dadurch besser lesbaren Version enthalten.

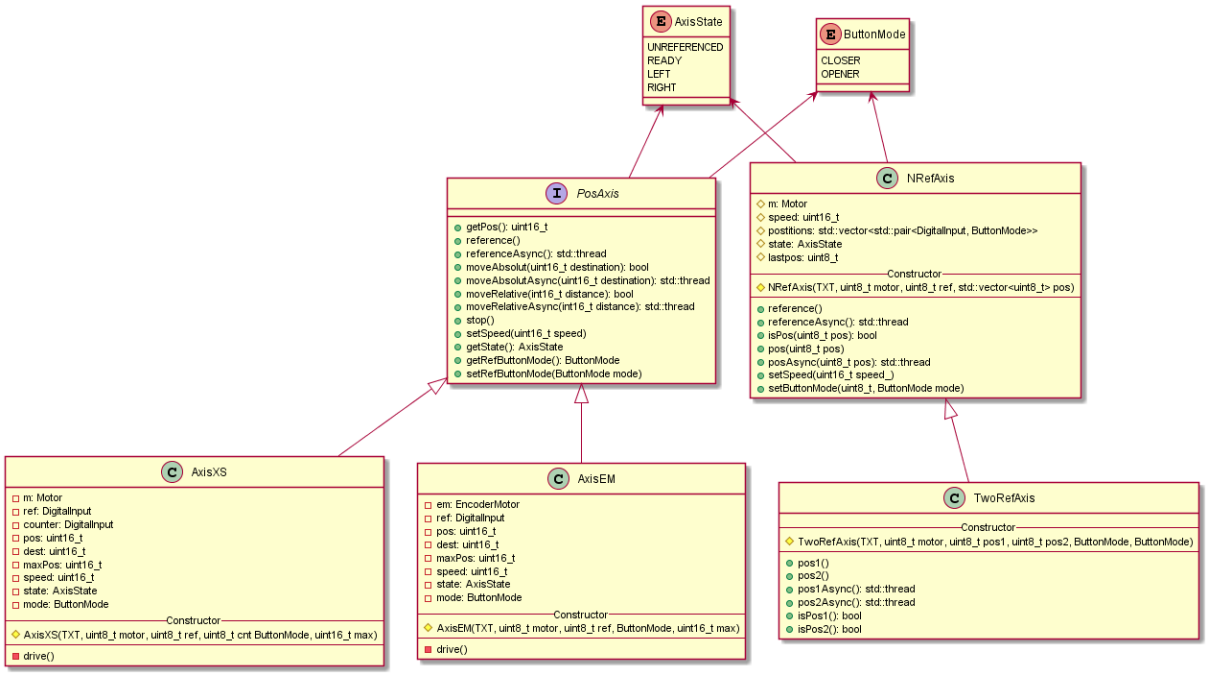


Abbildung 4.7: Klassendiagramm der Highlevel API

Insgesamt besteht die Highlevel API aus zwei Enumerationen und fünf Klassen. Das Interface *PosAxis* wird von den beiden Klasse *AxisXS* und *AxisEM* implementiert.

4.3 Utils API

Neben den beiden oben genannten Programmierschnittstellen zur direkten Ansteuerung der IO-Platine, sind in dieser API einige nützliche Funktionen enthalten, die zur Entwicklung von Programmen für den TXT-Controller sinnvoll sein können. Beim einbinden der Lowlevel API wird die Utils API automatisch mit eingebunden, da die Lowlevel API einige Funktionen daraus verwendet.

Die Utils API stellt vier Dinge bereit:

- Color Enumeration mit den Farbwerten *WHITE*, *RED*, *BLUE*
- eine Funktion, welche die Wartefunktion aus std::chrono vereinfacht: `sleep(1s)`
- eine Funktion, welche den vom Farbsensor ermittelten Wert in einen Wert der Color Enumeration konvertiert
- eine Funktion, welche den vom NTC ermittelten Widerstandswert in einen Temperaturwert konvertiert

```
1 TXT txt = TXT();
2 sleep(1s); //Wartezeit im Sekundenbereich
3 sleep(1ms);//Wartezeit im Millisekundenbereich
4 sleep(1us);//Wartezeit im Mikrosekundenbereich
5 Voltage v = txt.voltage(1);
6 Color c = convertToColor(v.value());
7 AnalogInput ai = txt.analogInput(2);
8 double temp = convertToTemperature(ai.value());
```

Listing 4.9: Verwendung von Funktionen aus der Utils API

Der Übergabeparameter der *sleep* Funktion zur Angabe der Wartezeit ist ein Objekt vom Typ `std::chrono::microseconds`. Objekte größerer Zeiteinheiten werden automatisch in Mikrosekunden umgewandelt.

Die Aufrufe von `convertToColor(value)` und `convertToTemperature(value)` müssen vom Entwickler normalerweise nicht aufgerufen werden, da die Klassen *ColorSensor* und *NTC*

bereits Methoden bereitstellen, die die konvertierten Werte zurückgeben. Es gibt trotzdem Anwendungsfälle, bei denen auf diese Funktionen zurückgegriffen werden muss.

Die Klassifikation von Farbwert (ist ein Spannungswert) zu einem Wert der Aufzählung *Color* in der Funktion `convertToColor` wird anhand eines einfachen Entscheidungsbaumes vorgenommen.

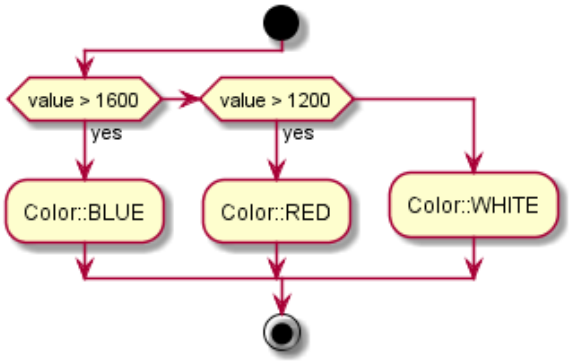


Abbildung 4.8: Entscheidungsbaum zur Farbklassifikation

Die Konvertierung des Temperaturwertes (Widerstandswert) zu einer Temperatur in der Grad-Celsius Skala wird anhand folgender Formel[3] durchgeführt:

$$temperatur = log(value) * log(value) * 1.3932 + log(value) * -43.942 + 271.87$$

[3]Konvertierungsformel aus der ROBOPro Bibliothek entnommen

5 Programmierung der Modellfabrik

In diesem Kapitel wird die neue Steuerung der fischertechnik Modellfabrik auf Basis von Lowlevel und Highlevel API vorgestellt. Der ist-Stand der Fabriksteuerung, insbesondere die Probleme und Schwierigkeiten dieser Steuerung werden in der Studienarbeit von Johannes Gaiser [Gai20] näher dargestellt.

In jedem der drei Abschnitte werden die Probleme der Steuerung der jeweiligen Fabrikeinheit zusammengefasst und Lösungen dafür aufgezeigt. Da sich die Hauptfunktionalität der einzelnen Einheiten nicht geändert hat, wird im folgenden nicht mehr auf den Ablauf der einzelnen Arbeitsschritte eingegangen. Es werden lediglich Optimierungen und Lösungen von Problemen angesprochen.

5.1 Sortiereinheit

In der bisherigen Implementierung der Sortiereinheit gibt es Probleme, wenn sich zwei oder mehr Werkstücke zur gleichen Zeit auf dem Fließband befinden. Der Sortiervorgang wird mit der Unterbrechung der ersten Lichtschranke gestartet und endet mit dem Einsortieren eines Werkstückes in eine Sortierbox. Wenn zwischen dem Passieren der ersten Lichtschranke und dem Einsortieren ein weiteres Werkstück die erste Lichtschranke durchläuft, wird dies nicht erkannt. Erst nachdem ein Werkstück das Fließband verlassen hat, darf das nächste die Sortiereinheit durchlaufen.

In der Neuimplementierung kann beim Passieren des Werkstückes der zweiten Lichtschranke das nachfolgende Werkstück in die Farberkennung einfahren. Dies wird gelöst, indem nach der Farberkennung ein weiterer Thread gestartet wird, welcher die Sortierung des Werkstückes vornimmt. Der aufrufende Thread kann bereits das nächste Werkstück entgegennehmen und bei diesem die Erkennung der Farbe vornehmen.

In diesem Erkennungsschritt zwischen den zwei Lichtschranken darf sich weiterhin nur ein Werkstück befinden, um Verwechslungen bei der Erkennung zu vermeiden.

Die Sortiereinheit besitzt sowohl für die Farberkennung als auch für die Sortierung der Werkstücke jeweils einen Zustand, der den Wert *WAITING* oder *WORKING* annehmen kann. Solange mindesten einer der beiden den Zustand *WORKING* annmmt, bewegt sich auch das Fließband.

5.2 Bearbeitungseinheit

An der Bearbeitungseinheit liegen im Beispielprogramm von fischertechnik keine schwerwiegenden Probleme vor. Deshalb ist die einzige Verbesserung in diesem Schritt eine geringere Ausführungszeit einiger Arbeitsschritte aufgrund von Parallelisierung. Mit der Highlevel API ist es jetzt sehr einfach Achsenfahrten parallel auszuführen, wovon die Steuerung der Bearbeitungseinheit bei der Referenzierung der Achsen sowie der Bewegung des kleinen Vakuumroboters und des Drehtisches einen zeitlichen Vorteil hat.

5.3 Hochregallager und Vakuumroboter

In der sowohl baulich als auch softwaretechnisch größten und komplexesten Teileinheit der Modellfabrik gibt es im Gegensatz zu den anderen beiden Einheiten deutlich mehr Verbesserungspotenzial. Da der Vakuumroboter und das Hochregallager aus jeweils mehreren Achsen bestehen, und diese wie im vorigen Abschnitt bereits erwähnt nicht parallel befahren werden, geht ein wesentlicher Zeitanteil für die sequenzielle Positionierung der Achsen verloren.

Als weiter kritisch anzusehenden Punkt absolvieren beide Roboter gelegentlich unnötige Achsenfahrten, die keinen funktionalen Vorteil bieten und dadurch wertvolle Laufzeit verschwendet wird.

Der Inhalt des Hochregallagers wird während des Betriebs nur temporär im Hauptspeicher gehalten und geht bei Abschaltung der Modellfabrik verloren. Ohne eine Persistenzmöglichkeit des Lagerinhaltes muss in das Hochregallager vor jedem Start der Modellfabrik manuell eingegriffen und dieses Leergeräumt werden, um Belegungskonflikte der Werkstücke zu vermeiden. Dies ist für eine Industrie 4.0 Anlage undenkbar.

Wie auch in der Bearbeitungseinheit ist die Parallelisierung der Achsenbewegungen mit der Highlevel API sehr einfach umzusetzen. Die zeitliche Ersparnis ist jedoch deutlich größer, da der größte Anteil der Aktionen von Hochregallager und Vakuumroboter Achsenoperationen sind. Zusätzlich zur Zeiteinsparung mit parallelen Achsfahrten kann auch Zeit durch Entfernen unnötiger Achsenbewegungen gespart werden. Insgesamt verringert sich der Gesamtdurchlauf der Modellfabrik um 5 Minuten, von 23 Minuten auf 18 Minuten, was nur noch ca. 78 % der ursprünglichen Bearbeitungszeit entspricht. Die größte Zeitersparnis fällt dabei auf die Parallelisierung der Achsen bei Hochregallager und Vakuumroboter.

Das Hochregallager und der Vakuumroboter zeichnen sich durch häufig ähnliche Arbeitsschritte aus. Um diese aus dem Hauptprogramm auszulagern und das Programm lesbarer zu gestalten, wurde für jeden der beiden Roboter eine Klasse entwickelt, welche für die häufigsten Anwendungsfälle Methoden bereitstellt.

Für die Persistierung des Inhaltes gibt es ein Interface *IStorage* mit der Implementierung *FileStorage*, die den Inhalt des Hochregallagers in einer Datei sichert. Der Lagerbestand wird in der Datei *lagerbestand.txt* auf dem TXT-Controller in einer Zeichenkette codiert abgelegt. Der Inhalt jeder Lagerposition ist in dieser Datei als einzelnes Zeichen codiert. Das Zeichen ’x’ repräsentiert eine Lagerposition ohne Lagerbox, ’e’ eine leere Lagerbox und die Zeichen ’w’, ’r’, ’b’ stehen für die Farben weiß, rot und blau. Mit Blick vom Hochregallager-Roboter auf das Hochregallager sind die Lagerpositionen von rechts nach links und von oben nach unten von 0 bis 8 durchnummeriert.

Das Hochregallager besitzt einen Status, der die folgenden Zustände annehmen kann:

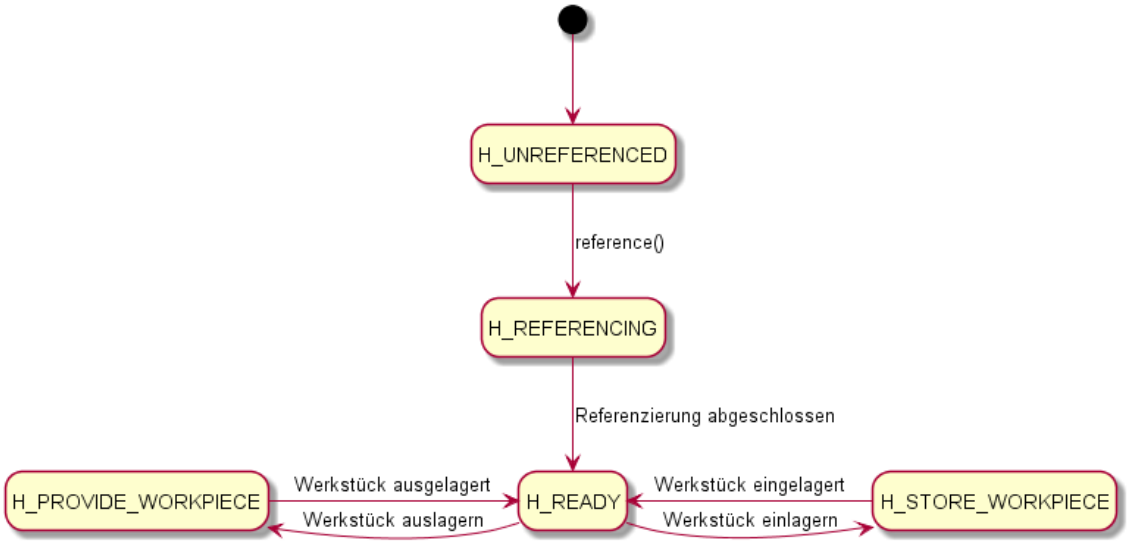


Abbildung 5.1: Zustandsdiagramm des Hochregallagers

Insgesamt sind für den Betrieb dieser Einheit die Lowlevel und Highlevel API sowie die Klasse *VacuumRobot*, *HighbayWarehouse*, *FileStorage* und die Enumerationen *WarehouseContent* und *HighBayState* notwendig. Durch Implementieren des Interfaces *IStorage* können anstatt der Speicherung des Lagerbestandes in einer Datei noch weitere Speichermöglichkeiten wie z.B. in einer Datenbank hinzugefügt werden. Die gesamte Klassenstruktur um Hochregallager und Vakuumroboter ist in Abbildung 5.2 dargestellt.

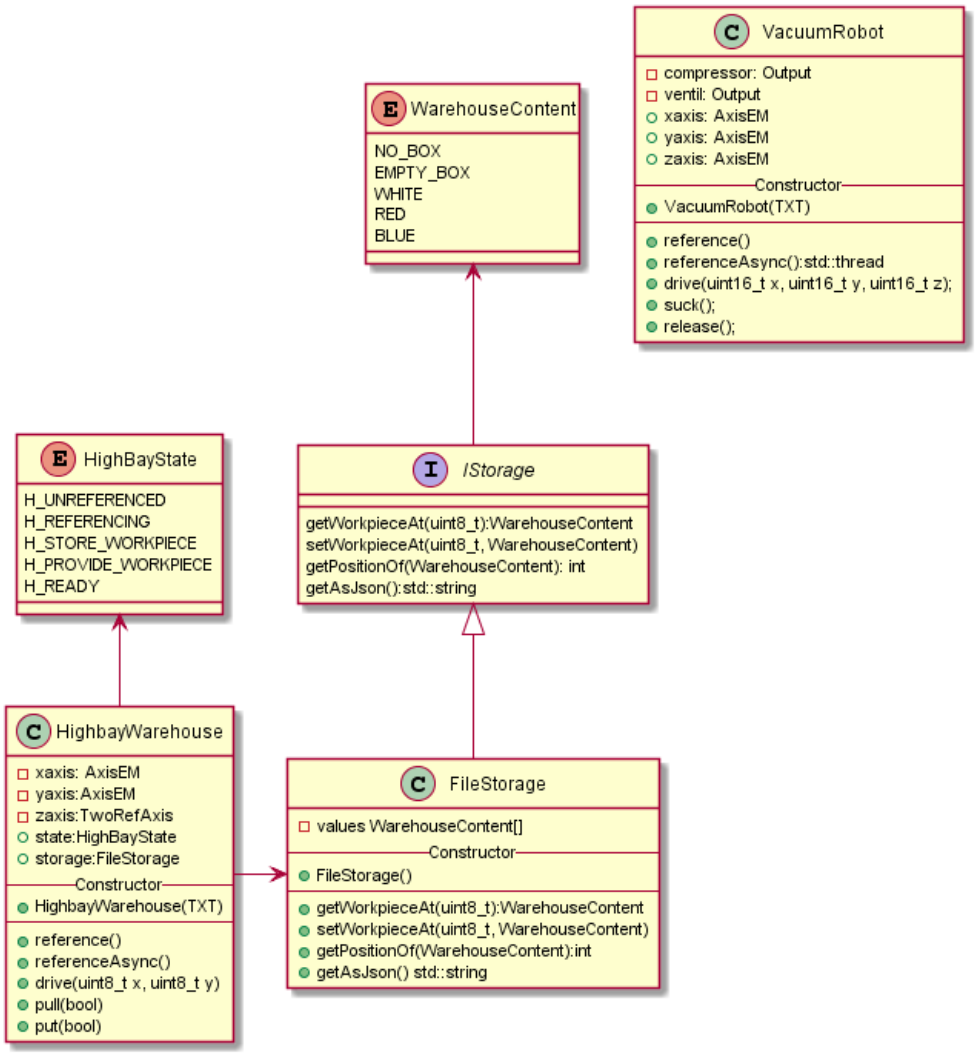


Abbildung 5.2: Klassendiagramm von Hochregallager und Vakuumroboter

6 Lehrbetrieb der Modellfabrik

Die fischertechnik Modellfabrik soll an der DHBW Stuttgart Campus Horb sowohl bei Werbeveranstaltungen im *Showbetreib* als auch für den regulären *Lehrbetrieb* genutzt werden. Im Lehrbetrieb sollen Teile der Fabriksteuerung von Studierenden im Rahmen von Laborübungen entwickelt werden. Dies ist durch häufiges Hochladen und Testen der Programme und Interaktion zwischen Studierenden und der Fabrik charakterisiert. Demgegenüber steht der Showbetrieb, bei dem die Fabrik selbstständig im Endlosmodus arbeitet und nicht auf eingreifen von Bedienern angewiesen ist. Im ersten Teilabschnitt dieses Kapitels wird ein Konzept vorgeschlagen, wie die Fabrik einfach zwischen diesen beiden möglichen Betriebsarten umgeschalten werden kann.

Die darauf folgenden Abschnitten betrachten die Betriebsart Lehrbetrieb näher. Dabei werden zwei mögliche Laborübungen im Rahmen der Modellfabrik, bzw. der Programmierung eines TXT-Controllers vorgestellt.

6.1 Backupkonzept zum Wechsel zwischen Show- und Lehrbetrieb

Der Wechsel der Modellfabrik zwischen den zwei Betriebsarten Lehrbetrieb und Showbetrieb soll für den Fabrikbediener einfach und in kurzer Zeit vorgenommen werden können. Neben dieser Anforderung soll der Wechsel außerdem vollständig am TXT-Controller vorgenommen werden können, ohne dass dazu ein weiterer Rechner benötigt wird.

Da die Modellfabrik sowohl im Lehrbetrieb als auch im Showbetrieb genutzt wird, ist die Empfehlung die Programme für den Showbetrieb in einem separaten Ordner (z.B. *Showbetrieb*) abzulegen, und nicht unter dem für Laborübungen gedachten Pfad

C-Program. Um ein versehentliches Löschen der Showbetriebprogramme über das Webinterface zu verhindern, sollen diese Programme nach dem Hochladen auf den TXT-Controller einmalig auf schreibgeschützt gestellt werden.

Dazu muss zunächst eine SSH-Verbindung zum TXT-Controller hergestellt werden. Dies ist in Unterabschnitt 3.1.2 beschrieben. Nach dem einloggen befindet sich der Bediener im Pfad */opt/knobloch*. Anschließend muss mit der Eingabe des Befehls `cd Showbetrieb` zum Verzeichnis *Showbetrieb* gewechselt werden. Um nun die darin enthaltene Steuerungsdatei auf schreibgeschützt zu stellen, muss der Befehl `chmod 400 <Dateiname>` ausgeführt werden.

Soll nun die Fabrik im Showbetrieb gestartet werden, muss der Bediener lediglich das Programm laden und starten. Dies geschieht über das übliche TXT-Controller Dateilademenü. Der TXT-Controller zeigt alle vorhandenen C/C++ Programme im Menü unter *C-Program* an, auch wenn diese in einem anderen Ordner abgelegt sind.

Für den Lehrbetrieb muss nichts Weiteres eingestellt werden. Die Studierenden laden ihre Programme mit der Webschnittstelle auf den TXT-Controller in den Ordner *C-Program* hoch und können diese auf gleichem Wege wie weiter oben bereits beschrieben laden und ausführen. Bei diesem Wechselkonzept zwischen Lehrbetrieb und Showbetrieb können die Studierenden die auf dem TXT-Controller hinterlegten Showprogramme so ebenfalls sehen und ausführen. Soll dies für Studierende nicht möglich sein, gibt es zwei möglich Alternativen, bei denen beim Wechsel der Betriebsart ein Rechner benötigt wird.

Die erste mögliche Alternative sieht dabei vor, dass die Showprogramme vor dem Beginn des Showbetriebs manuell über die Webschnittstelle hochgeladen werden und danach wieder gelöscht werden.

Bei der zweiten Alternative bleiben die Steuerungsprogramme dauerhaft auf dem TXT-Controller gespeichert und nur die Sichtbarkeit des Steuerungsprogramms wird gewechselt. Beim Wechsel von Showbetrieb auf Lehrbetrieb muss das Steurungsprogramm unsichtbar gemacht werden. Dazu wird eine SSH-Verbindung zum TXT-Controller benötigt. Unter Linux sind alle Verzeichnisse und Dateien mit einem '.' als Präfix im Namen unsichtbar. Es ist somit ausreichend die Steuerungsdatei mit dem Befehl `mv <Dateiname> .<Dateiname>` umzubenennen. Unsichtbare Programmdateien wer-

den vom TXT-Controller nicht angezeigt und können somit auch nicht zur Ausführung ausgewählt werden. Beim Wechsel von Lehrbetrieb nach Showbetrieb muss der umgekehrte Weg durchgeführt werden, um die Steuerungsprogramme wieder sichtbar zu machen.

Um bei technischem Defekt eines TXT-Controllers oder sonstigem Verlust eines der Steuerungsprogramme auf einem TXT-Controller die Steuerung wiederherzustellen, kann diese aus dem Git-Repository[1] dieser Arbeit entnommen werden. Die drei Programme für die Modellfabrik sind im Verzeichnis *factory/bin* enthalten. Unter Umständen müssen diese Steuerungsprogramme zuerst noch kompiliert werden, das benötigte Makefile liegt bei. Diese Programme können auch genutzt werden, um die Steuerungsprogramme für die Modellfabrik das Erste Mal auf die TXT-Controller zu laden.

6.2 Laborübung 1

Die erste Übung soll als Einstiegsübung betrachtet werden. Dabei wird davon ausgegangen, dass bis zu diesem Zeitpunkt die Bearbeiter der Übungsaufgabe nur geringe Programmierkenntnisse besitzen und noch keine Steuerung für einen fischertechnik TXT-Controller mit C++ entwickelt haben.

Diese Übung soll ein grundlegendes Verständnis über die Verwendung und den Aufbau der Lowlevel API vermitteln, deshalb sollen sowohl Eingänge als auch Ausgänge verwendet werden.

Die Übung besteht aus zwei kleinen Aufgaben. Im ersten Schritt ist von den Übenden eine Ampelsteuerung für ein Auto mit den Ampelphasen ROT, ROT-GELB, GRÜN, GELB zu entwickeln. Die Lösung dieser Aufgabe wurde in dieser Arbeit bereits an mehreren Stellen zum Vergleich der Programmiersprache herangezogen. Im zweiten Schritt ist eine Erweiterung dieser Ampelsteuerung um eine Fußgängerampel (ROT, GRÜN) sowie ein zugehöriger Taster vorgesehen. Die Autoampel bleibt so lange in der grünen Ampelphase, bis von einem Fußgänger mit dem Taster ein Signal zum Wechsel der Ampelphase angefordert wird. Anschließend schaltet die Autoampel über GELB auf ROT und die Fußgängerampel wird für eine gewisse Zeit grün. Nach der Grünphase der Fußgängerampel beginnt ein neuer Ampelzyklus.

[1] `https://github.com/SchmidJoel/FischertechnikTXTApi`

Für die Bearbeitung der Übungsaufgaben empfiehlt es sich den Übenden den Inhalt des gesamten Repositorys zur Verfügung zu stellen. Das Verzeichnis *exercises* enthält Vorlagen für die Übungsaufgaben, sowie ein fertiges Makefile für den Kompiliervorgang der Übungsaufgaben. Der Vorteil bei der Verwendung des gesamten Repositorys liegt darin, dass die benötigten Abhängigkeiten zur Entwicklung von Programmen mit den in dieser Arbeit vorgestellten APIs bereits vollständig Enthalten sind.

Da die Musterlösung dieser Aufgabe ebenfalls in diesem Repository enthalten ist, sollte diese gelöscht werden. Dies wird am einfachsten durch Entfernen des Verzeichnisses *examples*, welches die Musterlösung dieser Aufgabe beinhaltet sowie des Git-Verzeichnisses (*.git*) um die Wiederherstellung zu verhindern.

Bei Bedarf können die Übenden auf dem zur Verfügung gestellten Verzeichnis eine eigene Versionsverwaltung für die Versionierung der Übungsaufgaben einrichten. Dies ist bei dem Umfang der Übungsaufgaben vermutlich nicht notwendig.

Für die Bearbeitung der Übungsaufgaben muss außerdem die Web-Schnittstelle des TXT-Controller aktiviert werden (falls dies noch nicht der Fall ist).

Das Aufgabenblatt für diese Laborübung befindet sich in Abschnitt A des Anhanges.

6.3 Laborübung 2

Die zweite Übungsaufgabe kann unabhängig von der ersten bearbeitet werden, es werden aber mehr Programmierkenntnisse benötigt. In dieser Laborübung soll ein Programm für die Sortiereinheit der Modellfabrik mit folgendem Programmablauf entwickelt werden:

Das Fließband beginnt sich zu bewegen, sobald die Lichtschranke vor dem Farbsensor unterbrochen wird. Der Farbsensor führt so lange Messungen durch und speichert den niedrigsten Messwert, bis die Lichtschranke nach dem Farbsensor unterbrochen wird. Nun kann anhand des niedersten Messwertes die Farbe ermittelt werden. Das Werkstück bewegt sich weiter auf dem Fließband und wird je nach Farbe an unterschiedlichen Positionen vom Fließband in die Lagerbox geschoben. Das Fließband hört auf sich zu bewegen.

Nach der Programmieraufgabe gibt es noch eine optionale Zusatzaufgabe, in der gefragt wird, ob es zu Problemen kommt, wenn sich mehrere Werkstücke zeitgleich in der Sortiereinheit befinden.

Wenn bei einem Werkstück die Farbe gemessen wird und ein anderes Werkstück gerade sortiert wird, kann es je nach Implementierung der Sortiereinheit der Fall eintreten, dass nach dem Aussortieren eines Werkstückes das Fließband angehalten wird und das zweite Werkstück in der Farberkennung liegen bleibt. Dies könnte umgangen werden, indem die Farberkennung und die Sortierung als zwei separate Operationen angesehen werden. Solange eine der beiden Operationen ausgeführt wird, bewegt sich das Fließband, ansonsten hält es an. Es dürfen sich hier zwei Werkstücke gleichzeitig auf dem Fließband befinden, jeweils eines von ihnen in einer der genannten Operationen.

Die Musterlösung für die Programmieraufgabe befindet sich im Verzeichnis *factory* des Git-Repositorys. Dieses Verzeichnis sollte ebenfalls gelöscht werden, bevor das Repository den Übenden zur Verfügung gestellt wird.

7 Fazit

Zuerst wurde die historische Entwicklung der programmierbaren fischertechnik Controller vom ersten Computing Interface bis zur aktuellen Generation der TXT-Controller aus dem Jahr 2014 über die einzelnen Generationen betrachtet und deren schrittweise Verbesserung dargestellt.

Darauf aufbauend wurde eine Evaluation der möglichen Programmiersprachen für den TXT-Controller durchgeführt. Nach dem Vergleich der Sprachen bezüglich der beiden KO-Kriterien *Ansteuerung eines Extension-TXT* und *umfangreiche Bibliotheksunterstützung* blieb lediglich die Programmiersprache C/C++ für eine sinnvolle Weiterentwicklung der Modellfabrik im Industrie 4.0 Umfeld übrig. Aufgrund einiger Nachteile bei der Programmierung des TXT-Controller mit der von fischertechnik bereitgestellten Programmierschnittstelle für C/C++, wurden über diese weitere Abstraktionsebenen gebaut, mit dem Ziel diese Probleme zu beseitigen. Unter Verwendung von Lowlevel und Higlevel API wurde eine optimierte Steuerung der Fabrik für den Showbetrieb umgesetzt. Aufgrund paralleler Achsenbewegungen arbeitet die Modellfabrik mit dieser neuen Steuerung effizienter.

Auf Basis der entwickelten Lowlevel API wurden in dieser Arbeit abschließend zwei Laborübungen entworfen sowie ein Konzept zum Wechsel zwischen Show- und Lehrbetrieb ausgearbeitet.

Zusammenfassend kann gesagt werden, dass die angestrebten Ziele dieser Studienarbeit erfolgreich erreicht und umgesetzt werden konnten.

Glossar

Downloadbetrieb Ein auf den Controller/Interface geladenes Programm steuert die IO Pins, es ist keine Verbindung zu einem PC notwendig. 7, 16, 18, 32

Onlinebetrieb Die IO Pins des Controllers/Interfaces werden von einem PC über eine aktive Verbindung angesteuert. 6–8, 16, 18

Literatur

[Ylö96] Tatu Ylönen. „SSH - Secure Login Connections over the Internet“. In: *Sixth USENIX Security Symposium*. USENIX, 1996, S. 37–42.

[Mül05] Ulrich Müller. *Übersicht der Interfaces*. 2005. URL: https://www.ftcommunity.de/ftComputingFinis/interpd.html (besucht am 21. 03. 2020).

[Mül09] Ulrich Müller. *ROBO TX Controller 500995*. 2009. URL: https://www.ftcommunity.de/ftComputingFinis/txcontroller.html (besucht am 21. 03. 2020).

[Mül11a] Ulrich Müller. *Interfaces*. 2011. URL: https://www.ftcommunity.de/ftComputingFinis/sprachen.html (besucht am 16. 03. 2020).

[Mül11b] Ulrich Müller. *ROBO Interface 93 293*. 2011. URL: https://www.ftcommunity.de/ftComputingFinis/interrobo.html (besucht am 21. 03. 2020).

[Fou20a] Free Software Foundation. *GCC Cross Compiler*. 2020. URL: https://wiki.osdev.org/GCC_Cross-Compiler (besucht am 11. 04. 2020).

[Fou20b] Free Software Foundation. *GNU Make*. 2020. URL: https://www.gnu.org/software/make/ (besucht am 27. 03. 2020).

[Gai20] Johannes Gaiser. *Inbetriebnahme der Fischertechnik-9V-Modellfabrik mit Schwerpunkt Dokumentation und Prozessvisualisierung*. 2020.

[coma] fischertechnik community. *Computing Interfaces*. URL: https://ftcommunity.de/knowhow/computing/computing_interfaces/ (besucht am 10. 03. 2020).

[comb] fischertechnik community. *TX-Controller (2009)*. URL: https://ftcommunity.de/knowhow/computing/tx/ (besucht am 16. 03. 2020).

[comc] fischertechnik community. *TXT-Controller (2014)*. URL: https://ftcommunity.de/knowhow/computing/txt/ (besucht am 16. 03. 2020).

[fisa] fischertechnik. *Intelligent Interfaces*. URL: https://ftcommunity.de/knowhow/computing/intelligent_interface/intelligent_interface.pdf (besucht am 10. 03. 2020).

[fisb] fischertechnik. *Studium/ Berufsschule*. URL: https://www.fischertechnik.de/de-de/lehren/schulstufen/studium-berufsschule (besucht am 28. 04. 2020).

[mik] mikrocontroller.net. *Pulsweitenmodulation*. URL: https://www.mikrocontroller.net/articles/Pulsweitenmodulation (besucht am 26. 03. 2020).

Anhang

Im Anhang befinden sich in Abschnitt A ein Übungsblatt für die Programmierung einer Ampelsteuerung, in Abschnitt B wird die Sortiereinheit in einem zweiten Übungsblatt näher betrachtet und eine Steuerung für diese Entwickelt. Anschließend sind die beiden Klassendiagramme der Highlevel und Lowlevel API abgebildet.

A Übungsblatt 1

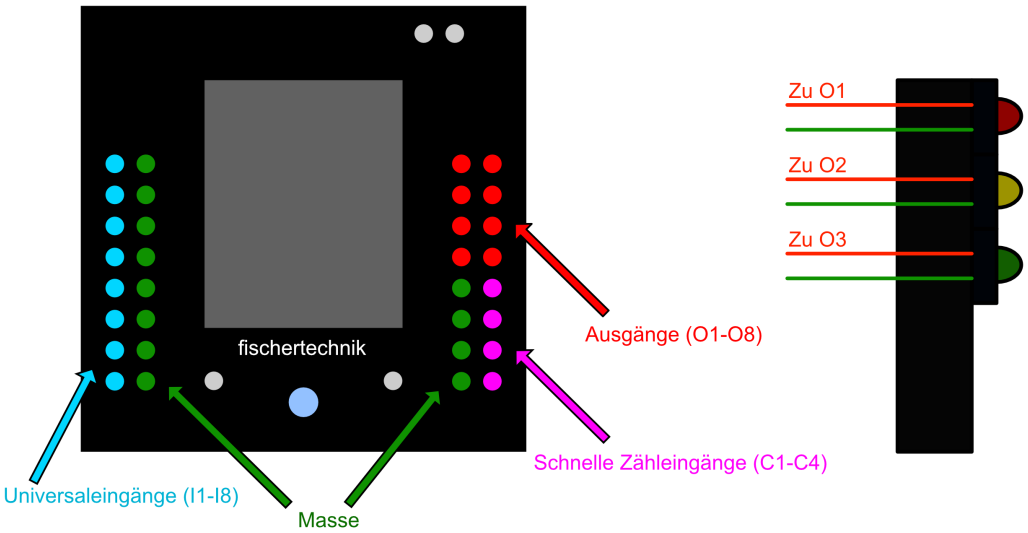
Mit diesem Übungsblatt lernen Sie die Programmierung des fischertechnik TXT-Controller in der Programmiersprache C++. Das Ziel dieses Aufgabenblattes ist das Erkennen von einfachen Eingaben sowie die Ansteuerung von Ausgängen am TXT-Controller.

Für die Bearbeitung dieses Aufgabenblattes benötigen Sie einen TXT-Controller, 5 Lampen einen Taster sowie ein USB-Kabel und 6 Verbindungskabel um die Lampen bzw. den Taster mit dem TXT-Controller verbinden zu können. Zusätzlich können Sie zwei rote, zwei grüne und eine gelbe Abdeckung für die Lampen und schwarze Bausteine nutzen, um den Aufbau realitätsnaher zu gestalten.

In den folgenden zwei Aufgaben lernen Sie nun Schritt für Schritt die Programmierung des TXT-Controllers am Beispiel einer Ampelsteuerung.

A.1 Ampelsteuerung

Bevor Sie mit dem Zusammenbau und der Programmierung der Ampel beginnen, betrachten wir gemeinsam den Aufbau des fischertechnik TXT-Controller:



Auf der linken Seite befinden sich acht Universaleingänge, die für den Anschluss von Sensoren (z.B. Fototransistor, Taster, Farbsensor, Abstandssensor, ...) genutzt werden

können. Daneben befinden sich acht weitere Anschlüsse, an denen Masse anliegt. Alle Sensoren benötigen einen Pin der Universaleingänge (I1-I8) und einen Masseanschluss. Auf der rechten Seite des Displays befinden sich die Anschlüsse für die normalen Ausgänge (O1-O8). An ihnen werden beispielsweise Lampen angeschlossen, dazu wird ein Anschluss der Lampe an einen O-Ausgang des TXT und der andere an einen Massepin angeschlossen. Alternativ können zwei Ausgänge auch kombiniert als ein Motorausgang genutzt werden. Außerdem befinden sich auf der rechten Seite Anschlüsse für vier schnelle Zähleingänge, um Schrittmotoren zu steuern.

Bauen Sie als Erstes die Ampel zusammen und schließen Sie diese wie in der Abbildung gezeigt an.

Öffnen Sie nun in einem Editor Ihrer Wahl die Datei *Aufgabe1.cpp*. Dies ist ein bereits lauffähiges Programm für den TXT-Controller. Führen Sie mit dem Befehl `make Aufgabe1` (UNIX-Betriebssystem) bzw. `mingw32-make Aufgabe1` (Windows) das Makefile für dieses Programm aus. Das Programm wird nun in Maschinencode übersetzt, welcher vom TXT-Controller ausgeführt werden kann. Öffnen Sie im Browser die Adresse 192.168.7.2 und geben für den Benutzernamen *TXT* und als Passwort die Nummer, die in der Statusleiste des TXT-Displays zu sehen ist ein. Über diese Oberfläche können Sie Programme auf den TXT-Controller laden. Wechseln Sie in der Web-Oberfläche in das Verzeichnis *C-Program* und laden Sie ihr übersetztes Programm hoch. Sie finden es im Unterverzeichnis *bin* des Ordners, welches auch Ihren Programmcode enthält.

Nachdem Sie das Programm auf den TXT-Controller hochgeladen haben, müssen Sie es vor dem Starten zuerst laden. Dies wird über das Display des TXT-Controllers gemacht. Da Ihr Programm noch keinen Programmcode enthält, beendet es sich nach dem Start sofort wieder.

Fügen Sie nun eine weitere Zeile unter `TXT txt = TXT();` ein: `Output lampe = txt.output(1);`. Mit dieser Zeile sagen Sie dem TXT-Controller, dass am Anschluss O1 eine Lampe angeschlossen ist, je nachdem welche Zahl sie in den Klammern angeben wird ein anderer Ausgang gewählt. Die Lampe würde nach einem weiteren Übersetzungsvorgang aber noch nicht leuchten. Um die Lampe anzuschalten, ergänzt man das Programm noch um die Zeile: `lampe.on();`. Die Lampe leuchtet jetzt zwar kurz (nicht Sichtbar), geht aber sofort wieder aus, da das Programm wieder beendet wird. Sie können mit

`sleep(1s);` eine Wartezeit von einer Sekunde einbauen.

Jetzt haben Sie alle notwendigen Kenntnisse um eine Ampel für einen einzigen Zyklus zu programmieren. Die Ampelphasen sollen in der Reihenfolge rot, rot-gelb, grün, gelb ausgeführt werden. Zwischen den einzelnen Ampelphasen soll eine kurze Wartezeit eingebaut werden.

In dieser Übungsaufgabe werden sehr viele Schritte durch die Schnittstelle *TXT_lowlevel_API.h* vereinfacht. Diese reicht die Befehle im Hintergrund an die fischertechnik Programmierschnittstelle des TXT-Controllers weiter. Beim Aufruf der Methode `on()` auf ein Output-Objekt wird im Hintergrund eigentlich dieser Befehl ausgeführt: `pTArea->ftX1out.duty[0] = 512;` - der Ausgang O1 bekommt den PWM-Wert 512 zugewiesen und leuchtet dadurch mit voller Helligkeit.

A.2 Fußgängerampel

Erweitern Sie die in Aufgabe 1 entwickelte Ampelsteuerung um eine Fußgängerampel mit den Ampelphasen rot und grün sowie einem Taster, mit dem ein Fußgänger dem TXT-Controller mitteilen kann, dass er gerne die Straße überqueren möchte. Die Autoampel bleibt so lange in der grün-Phase, bis der Taster gedrückt wird. Nach einer kurzen Wartezeit wird die Ampel in rot-Phase geschalten und die Fußgänger bekommen für 5 Sekunden ein grünes Signal. Anschließend beginnt der Ampelzyklus von vorne. Mit `DigitalInput taster = txt.digitalInput(1);` teilen Sie dem TXT-Controller mit, dass am Eingang I1 ein Taster angeschlossen ist. Mit `taster.waitFor(DigitalState::HIGH);` wartet der TXT-Controller solange mit der weiteren Programmausführung, bis der Taster gedrückt wird.

B Übungsblatt 2

Eine der drei Einheiten der Modellfabrik ist die Sortiereinheit, diese ist dafür verantwortlich, dass die Werkstücke anhand ihrer Farbe in das richtige Fach einsortiert werden und der Vakuumroboter sie von dort abholen kann. In dieser Übung werden Sie die Sortiereinheit der Modellfabrik entwickeln. Der Ablauf soll folgendermaßen aussehen:

Das Fließband beginnt sich zu bewegen, sobald die Lichtschranke (I1) vor dem Farbsensor unterbrochen wird. Der Farbsensor führt so lange Messungen durch und speichert den niedrigsten Messwert, bis die Lichtschranke nach dem Farbsensor unterbrochen wird. Nun kann anhand des niedersten Messwertes die Farbe ermittelt werden. Das Werkstück bewegt sich weiter auf dem Fließband und wird je nach Farbe an unterschiedlichen Positionen vom Fließband in die Lagerbox geschoben. Das Fließband hört auf sich zu bewegen.

Bauteil	**Anschluss**	**Bauteil**	**Anschluss**
Lichtschranke 1	I1	Zylinder weiß	O5
Farbsensor	I2	Zylinder rot	O6
Lichtschranke 2	I3	Zylinder blau	O7
Motor	M1	Kompressor	O8
Impulstaster	C1		

B.1 Programmierung

Entwickeln Sie eine Steuerung für die Sortieranlage mit dem oben genannten Ablauf. Als Vorlage für dieses Programm können Sie die Datei *Aufgabe2.cpp* nutzen. Die Funktion `ConvertToColor(farbwert)` kann eine Konvertierung des Farbwertes in eine Farbe vornehmen. Von der zweiten Lichtschranke bis zur weißen Auslagerstelle müssen 6 Impulse zurückgelegt werden, für rot 16 und für blau 26.

B.2 Zusatzaufgabe

Was passiert in ihrem Programm, wenn sich zeitgleich mehrere Werkstücke auf dem Fließband befinden? Wie kann dieses Problem behoben werden?

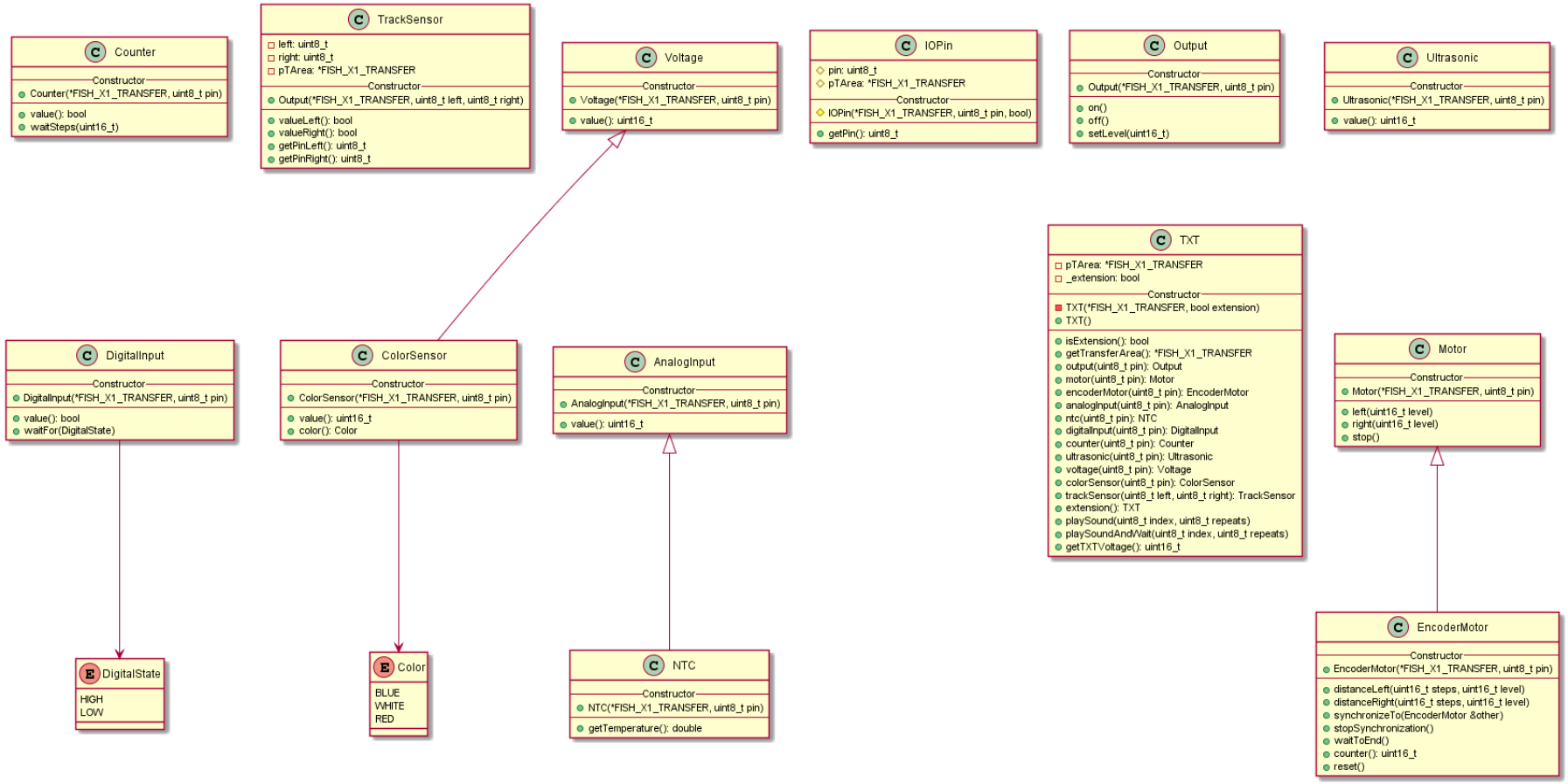


Abbildung 8.1: Klassendiagramm Lowlevel API

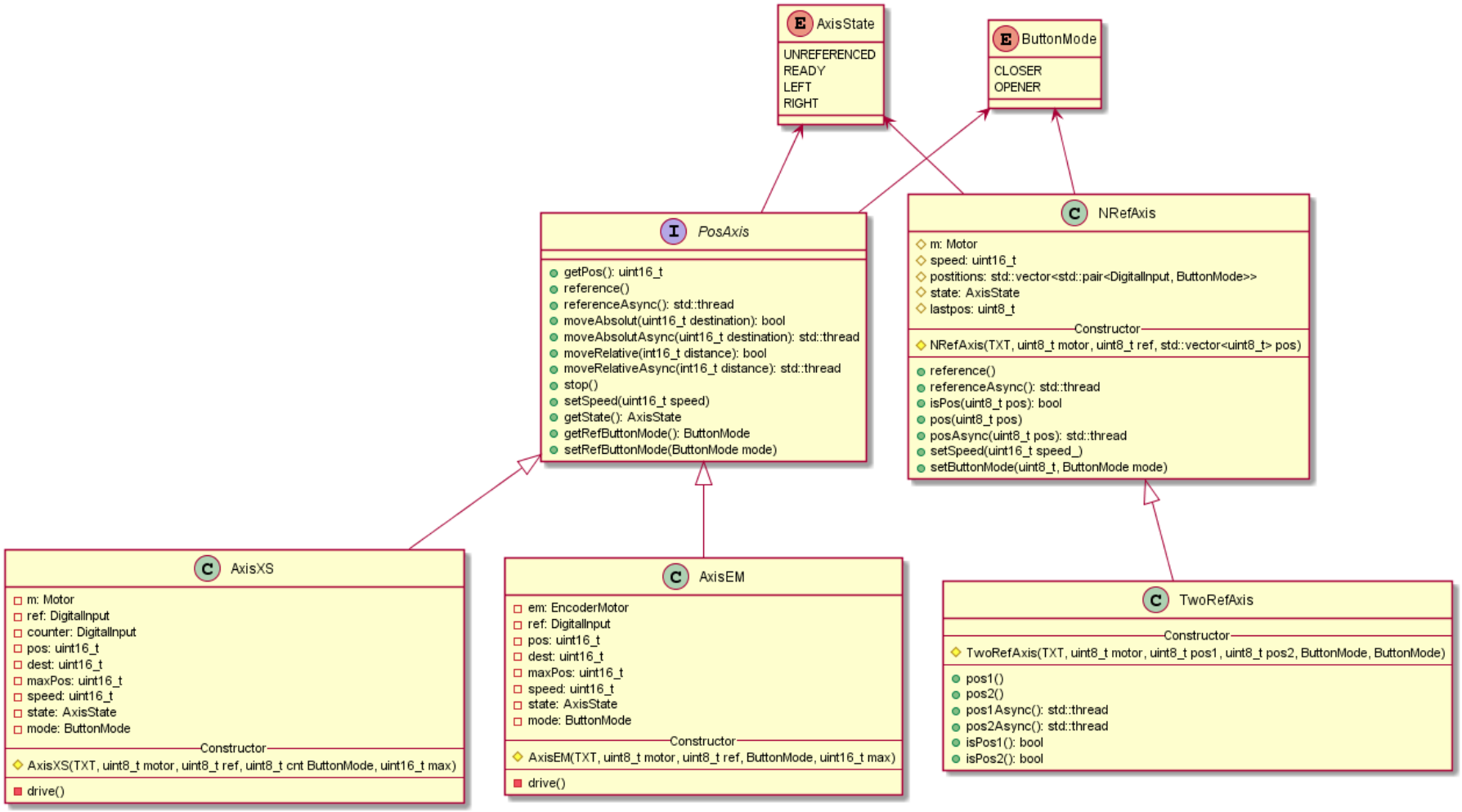


Abbildung 8.2: Klassendiagramm Highlevel API