



Hochschule für Technik,  
Wirtschaft und Kultur Leipzig

BELEGARBEIT IM FACH EMBEDDED SYSTEMS III

PRÜFER:

PROF. DR. ANDREAS PRETSCHNER

---

**Steuerung einer  
SPS-Automatisierungsanlage durch ein  
STM32-Board**

---

Vorgelegt von: EMANUEL ECKSTEIN  
Studiengang: ELEKTRO- UND INFORMATIONSTECHNIK  
Seminargruppe: 23-EIM AT  
Matrikelnummer: 85117  
E-Mail-Adresse: EMANUEL.ECKSTEIN@STUD.HTWK-LEIPZIG.DE

Vorgelegt von: TOBIAS RAUCH  
Studiengang: ELEKTRO- UND INFORMATIONSTECHNIK  
Seminargruppe: 23-EIM AT  
Matrikelnummer: 78309  
E-Mail-Adresse: TOBIAS.RAUCH@STUD.HTWK-LEIPZIG.DE

11. Oktober 2024

# Inhaltsverzeichnis

<b>1 Hardware und Systemaufbau</b>	<b>1</b>
1.1 STM32 MP135F Board . . . . .	1
1.1.1 Boardspezifikationen . . . . .	2
1.1.2 Anschlussperipherie . . . . .	3
1.2 Automatisierungsanlage Sortierer-1000 . . . . .	4
1.2.1 mechanische Bauteile . . . . .	4
1.2.2 elektronische Bauteile . . . . .	5
1.2.3 weitere Bauteile . . . . .	5
1.3 Schnittstellen . . . . .	6
<b>2 Methodik und Prozessimplementierung</b>	<b>7</b>
2.1 Inbetriebnahme des STM32-Boards . . . . .	7
2.2 Paketauswahl und Bau des Betriebssystems . . . . .	7
2.2.1 Paketauswahl . . . . .	8
2.2.2 Bau des Images und Installation Python Pakete . . . . .	10
2.3 Prozessablauf der Automatisierungsanlage . . . . .	11
2.4 Projektcode . . . . .	14
<b>3 Fazit</b>	<b>15</b>
3.1 Funktionstest . . . . .	15
3.2 Ausblick . . . . .	15
<b>Abbildungsverzeichnis</b>	<b>16</b>
<b>Anhang</b>	<b>18</b>

# 1 Hardware und Systemaufbau

In diesem Kapitel soll die verwendeten Hardware vorgestellt werden. Im Vordergrund steht dabei das STM-Board und die Automatisierungsanlage sowie die entsprechende Schnittstelle.

## 1.1 STM32 MP135F Board

Bei dem verwendeten STM32MP135F-DK handelt es sich um ein Embedded Development Kit, welches zum Beispiel ideal für Prototypenentwicklung, industrielles Internet of Things (i-IoT), eingebettete Linux-Anwendungen, Human-Machine Interfaces (HMIs) und andere Projekte, bei denen eine Kombination aus komplexen Betriebssystemaufgaben und Echtzeitsteuerung gefordert wird, ist. Es wird allgemein gesagt von Entwicklern verwendet, um Hardware- und Softwareprojekte zu testen, Prototypen zu bauen oder neue Controller/Prozessoren/Sensoren zu evaluieren.

### 1.1.1 Boardspezifikationen

Folgende wichtige Mikrocomputer-Board Komponenten sind auf dem STM32MP132F-DK2 verbaut:

- Mikroprozessor STM32MP135FAF7 MPU

*Der STM32MP135FAF7 MPU ist ein leistungsfähiger Mikroprozessor mit einem ARM Cortex-A7 Kern, der für die Ausführung von Betriebssystemen und komplexen Anwendungen optimiert ist. Die Taktfrequenz beträgt bis zu 1 GHz, was eine hohe Rechenleistung für verschiedene Embedded-Anwendungen bietet. Das TFBGA320-Gehäuse ist ein sehr kompaktes Paket mit 320 Anschlusspunkten. Der Prozessor bietet eine hohe Rechenleistung und Energieeffizienz, die sich gut für Betriebssysteme und ressourcenintensive Anwendungen eignet.*

- Stromversorgungs-IC ST PMIC STPMIC1

*Der STPMIC1 ist ein Power Management IC (PMIC) von STMicroelectronics, der speziell für die Versorgung von Mikroprozessoren wie der STM32MP1-Serie (einschließlich STM32MP132) entwickelt wurde. Er bietet eine umfassende Lösung zur Verwaltung der Stromversorgung verschiedener Spannungsbereiche, die für Mikroprozessoren und deren Peripherie erforderlich sind. Außerdem bietet er verschiedene Spannungsregler, Schutzmechanismen und ein dynamisches Power Management, um den Energieverbrauch zu optimieren und die Lebensdauer von batteriebetriebenen Geräten zu verlängern.*

- 512MB Arbeitsspeicher

*Der verbaute DDR3-L RAM-Chip ist auf 533 MHz getaktet*

- 4x Anzeige-LEDs

- 2x Benutzereingabe-Knöpfe, 1x Energiesparmodus-Knopf, 1x Tamper

- 4.3"480×272 LCD-Dispaly mit Berührungssteuerung

- Bluetooth 4.1 und WiFi 802.11b/g/n

### **1.1.2 Anschlussperipherie**

Auch eine Vielzahl von physikalischen Ports sind auf dem Board verbaut:

- 1-Gbit/s Ethernet RJ45
- 15W USB-C Stromversorgung
- microSD-Kartenfach
- GPIO-Peripherie nach Raspi-Aufbau
- Flachband Bildschirmanschluss
- 4x USB-A, 1x USB-C Host
- HDMI (normal)
- STLINK-V3E Debugger
- Headset-Klinkenport

## 1.2 Automatisierungsanlage Sortierer-1000

Bei der Automatisierungsanlage geht es um eine sensorbasierten Sortiermaschine. Diese wurde im voraus unter der Leitung von Herrn M.Sc. Marco Braun zusammengebaut. Zusammengefasst wird ein Gegenstand aus einem Vorratsmagazin auf das Förderband geschoben und gestempelt, und, je nach Sensormessung, entweder in das Ziellager für nicht-metallische Teile, oder für metallische Teile befördert.

### 1.2.1 mechanische Bauteile

- Kolben zum Schieben auf das Förderband
- pneumatischer Stempel
- Vorratsmagazin
- metallisches Ziellager
- nicht-metallisches Ziellager
- pneumatischer Abstreifer
- Förderband

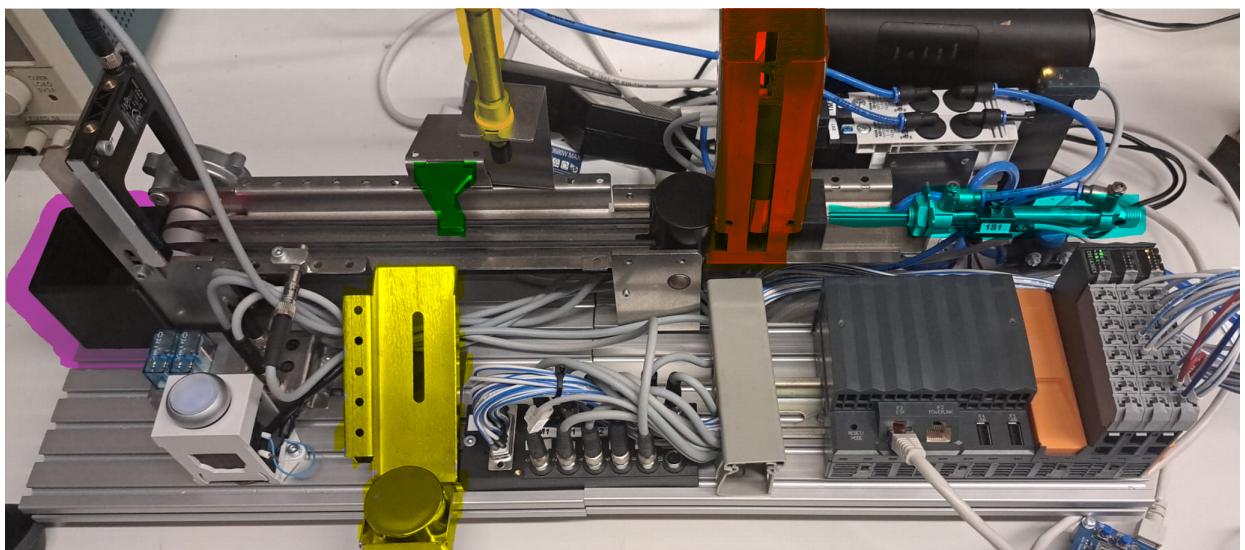


Abbildung 1.1: Automatisierungsanlage, mechanische Bauteile hervorgehoben

### 1.2.2 elektronische Bauteile

- Elektromotor des Förderbands
- Induktionssensor
- SPS (*B&R X20CP1382*)
- An-Schalter
- Lichtschranke
- 24V Netzteil

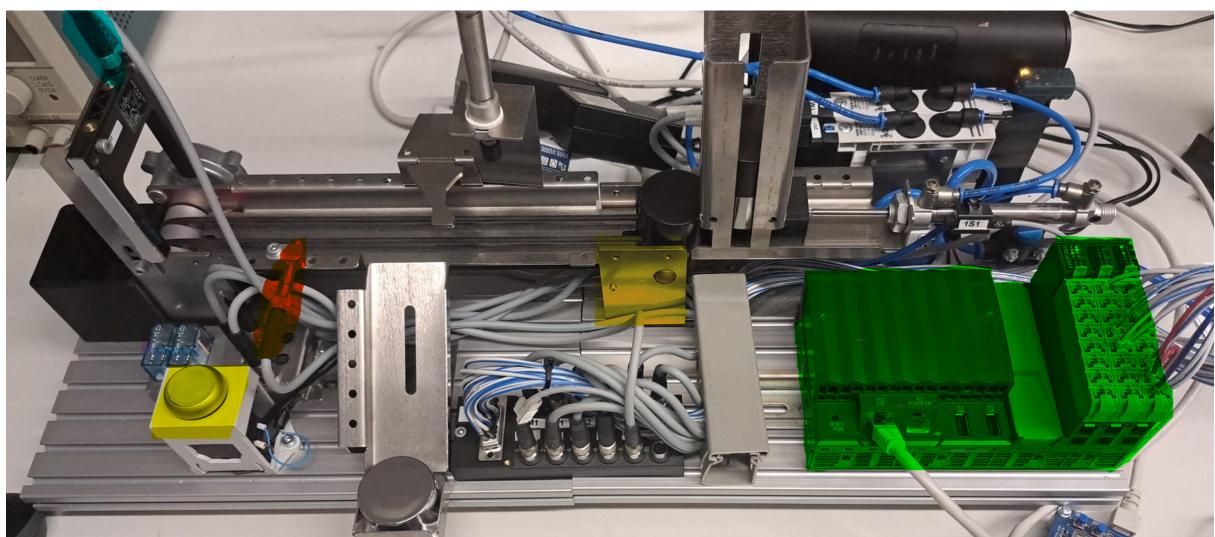


Abbildung 1.2: Automatisierungsanlage, elektrische Bauteile hervorgehoben

### 1.2.3 weitere Bauteile

Unter der Bezeichnung weitere Bauteile sollen alle sonstigen Komponenten der Pneumatik, Elektronik und Statik zusammengefasst werden. Sie dienen der Funktionalität, sind aber nur in ihrer Gesamtheit wichtig und sollen im Rahmen vom diesem Beleg nicht einzeln erwähnt werden.

## **1.3 Schnittstellen**

Die zentrale Schnittstelle wird über die Verbindung zwischen der SPS und dem Entwicklerboard definiert. Beide sind über ein Ethernet-Kabel mit demselben Router verbunden. Dieser gibt jedem Gerät eine IP-Adresse. Darüber kann nicht nur das Board die SPS ansteuern, sondern es können auch mit einem weiteren Entwickler-PC neue Software-Iterationen getestet werden.

# 2 Methodik und Prozessimplementierung

In diesem Kapitel sollen die im Projekt verwendeten Methoden betrachtet werden, vom blanken STM32-Board bis zur Automatisierungsanlage.

## 2.1 Inbetriebnahme des STM32-Boards

Das STM32-Board wurde nach dem vorgegebenen Beleg (von Finn Engler und Jeremias Seidler) und der Vorlesung nach in Betrieb genommen. Eine Ausnahme hierbei bildet der bitbake Befehl. Hier wurde statt dem vorgegebenen Befehl:

```
1 bitbake st-image-core
```

die Variante des

```
1 bitbake st-image-weston
```

verwendet. Das hat den Hintergrund das das Board somit eine einfache Wayland Unterstützung bietet, welche für das verbaute Display nötig ist.

## 2.2 Paketauswahl und Bau des Betriebssystems

In dem Projekt solle eine Python Applikation auf dem STM32 in Betrieb genommen werden. In dem Programm gibt es folgende Bibliotheken welche ebenfalls benötigt werden: OPCUA und PythonGTK.

Um diese Applikation in Betrieb zu nehmen müssen in der local.conf Datei einige Rezepte zum Bau des Images vorgenommen werden.

## 2.2.1 Paketauswahl

### Größe des Betriebssystems

Im ersten Schritt wurde die maximale Größe des Rootfiles vergrößert um eventuelle Speicherplatzprobleme zu beheben. Hierbei wird die Größe auf 1024 MB eingestellt. Die erfolgt mit der folgenden Zeile:

```
1 IMAGE_ROOTFS_MAXSIZE = "1024000"
```

### Python

Um die Applikation, welche auf dem Board laufen soll nutzen zu können müssen folgende Python Rezepte in der Datei angegeben werden:

```
1 IMAGE_INSTALL_append = " python3 python3-pip python3-requests  
    python3-numpy python3-dev python3-setuptools python3-  
    pillow python3-lxml"
```

Eine kurze Erklärung zu den einzelnen Python Rezepten:

- **python3:** Wird benötigt, für die Python3-Laufzeitumgebung.
- **python3-pip:** Der Paketmanager für Python, um zusätzliche Python-Pakete zu installieren.
- **python3-requests:** Eine beliebte Bibliothek für HTTP-Anfragen.
- **python3-numpy:** Eine Bibliothek für numerische Berechnungen. (optional)
- **python3-dev:** Die Entwicklungsheader für Python, die beim Kompilieren von C-Erweiterungen benötigt werden.

- `python3-setuptools`: Ein Paket zur Verwaltung von Python-Paketen und deren Abhängigkeiten.
- `python3-pillow`: Eine Bibliothek zur Bearbeitung und Anzeige von Bildern. (optional)
- `python3-lxml`: Eine leistungsstarke Bibliothek zur Bearbeitung von XML und HTML (optional).

Einige Pakete wurden installiert um andere Grafikumgebungen zu testen und in Betrieb zu nehmen. Leider hat dies nicht funktioniert weshalb wir uns für GTK entschieden haben.

## OPC UA

Ebenfalls um OPCUA nutzen zu können müssen der `gcc` und der `g++` Compiler mit als Rezept angegeben werden:

```
1 IMAGE_INSTALL:append = " gcc g++"
```

Für eine sichere Verbindung der für die Kommunikation herzustellen werden folgende Rezepte hinzugefügt:

```
1 IMAGE_INSTALL:append = " openssl libssl libffi"
```

## Zusätzliche Pakete

Um kleine Änderungen am Programmcode vorzunehmen wurde der Editor `Nano` ebenfalls als Rezept hinzugefügt:

```
1 IMAGE_INSTALL:append = " nano"
```

Durch einige Versuche mit anderen grafischen Oberflächen welche eine GUI in Python zur Verfügung stellen wurde folgende Pakete hinzugefügt, welche allerdings für die aktuelle Version unseres Setups nicht benötigt werden:

```
1 IMAGE_INSTALL:append = " libxml2-dev libxslt"
```

```
2 IMAGE_INSTALL:append = " libsdl2 libsdl2-image libsdl2-mixer  
    libsdl2-ttf"
```

Diese Pakete werden benötigt, wenn SDL Anwendungen wie PyGame oder Kivy für eine grafische Oberfläche benutzt werden.

## 2.2.2 Bau des Images und Installation Python Pakete

Das Image wurde anschließen mit dem Befehl

```
1 bitbake st-image-weston
```

gebaut. Danach wurde das erstellte Image über den STM32Cube-Programmer auf das Board überspielt.

Nach dem der Bootprozess abgeschlossen ist, kann man sich über eine angeschlossene Tastatur oder per SSH-Zugriff die benötigten Bibliotheken für die Python Applikation installieren. Allerdings muss hierzu das Board mit einem Ethernet-Kabel verbunden werden, da die Packete aus dem Internet gedownloaded werden.

Für diese Applikation werden die Python Bibliotheken OPC UA und GTK benötigt. Diese werden mit folgenden Befehlen auf dem Board installiert:

```
1 pip3 install opcua PyGObject
```

Durch den Bau des Weston-Images welches eine einfache Wayland Unterstützung liefert, war in diesem Fall die Bibliothek GTK bereits auf dem STM32 installiert und musste nicht extra installiert werden.

## 2.3 Prozessablauf der Automatisierungsanlage

Wie bereits in 1.2 erklärt, ist es das Ziel der Anlage gleichförmige Objekte in metallisch und nicht-metallisch mittels eines Induktionssensors zu sortieren. Die konkrete Ablaufsteuerung wird durch dieselben Bilder dargestellt, die später dann auf dem 4.3" Display gezeigt werden. Die Fallunterscheidung findet im vorletzten Zustand **CHECK** statt, nachdem entweder in das metallische (2.7 **OUTSOURCE**) oder die nicht-metallische (2.6 **FREE**) Teilelager sortiert wird.

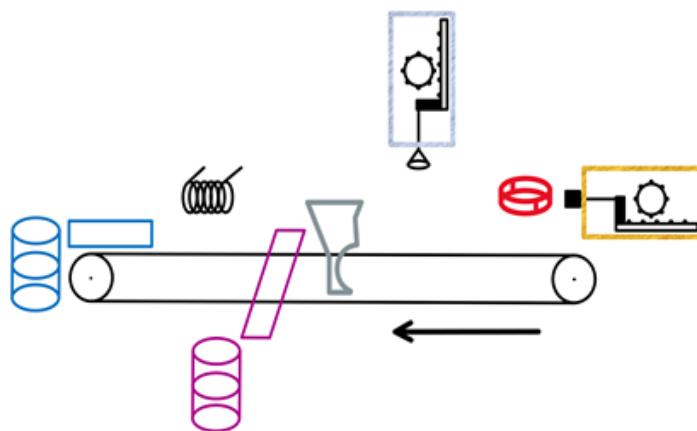


Abbildung 2.1: INIT

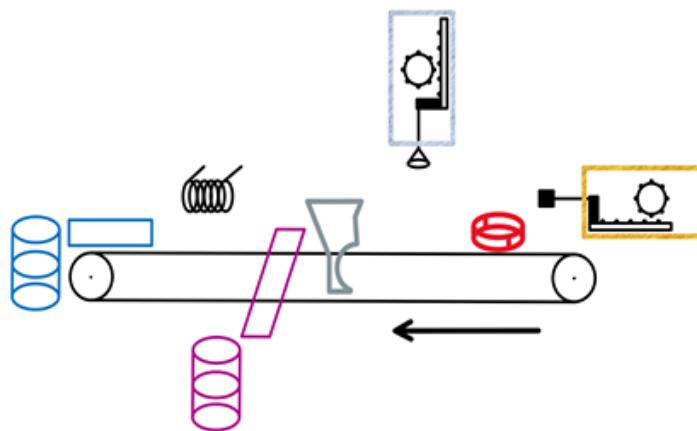


Abbildung 2.2: PRESS\_OUT

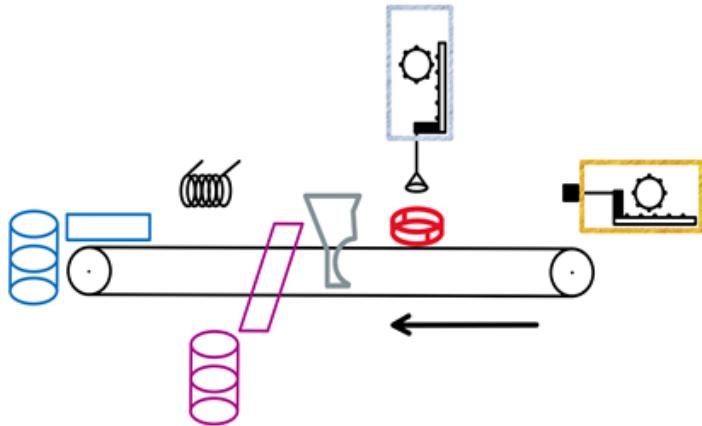


Abbildung 2.3: STAMP

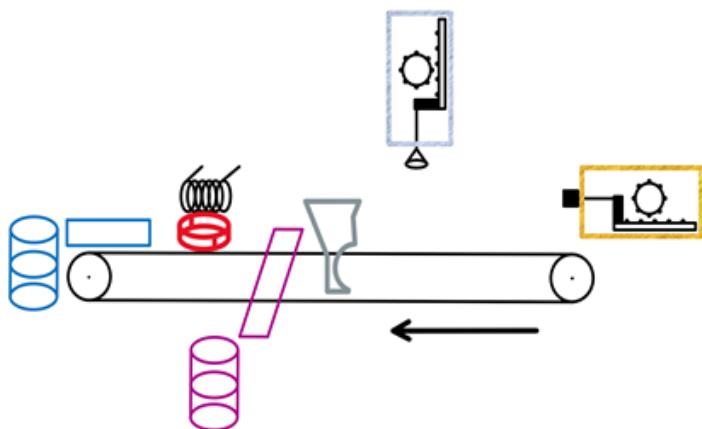


Abbildung 2.4: CHECK

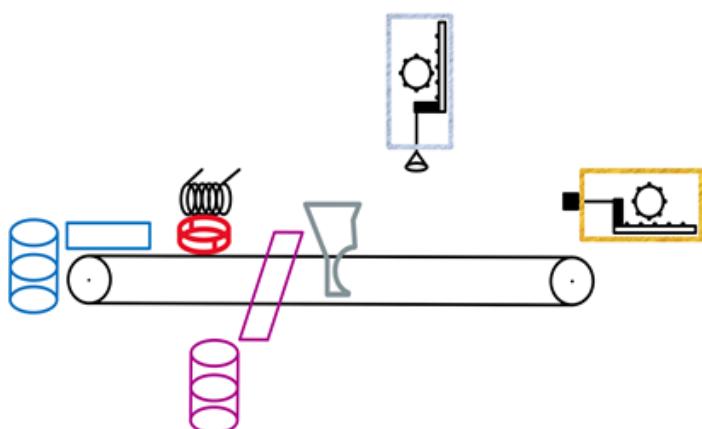


Abbildung 2.5: STOP

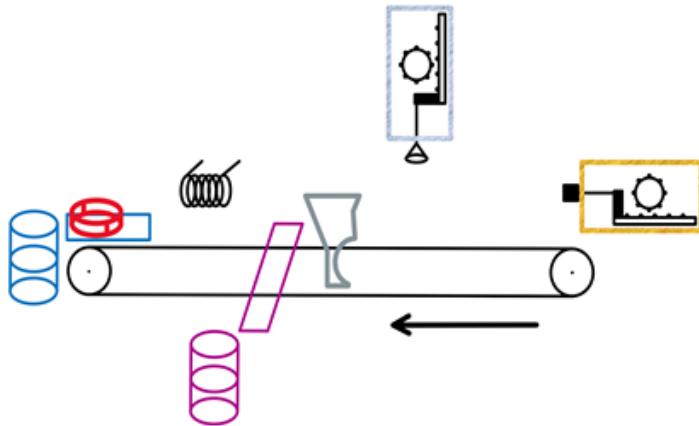


Abbildung 2.6: nicht-metallisch: FREE

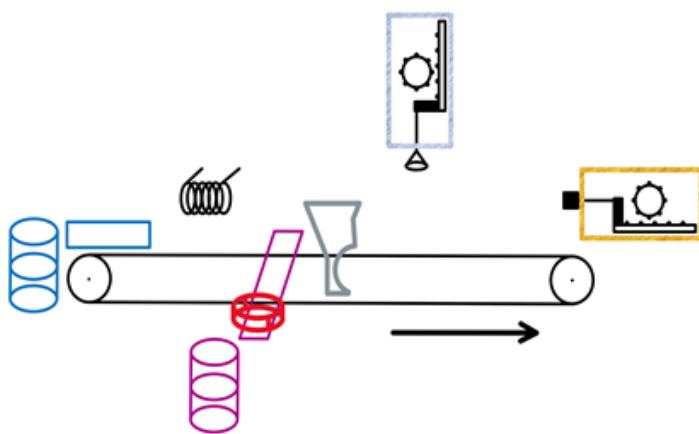


Abbildung 2.7: metallisch: OUTSOURCE

## 2.4 Projektcode

Der Steuerungscode der SPS für die Automatisierungsanlage wurde in Structured Text (ST) geschrieben und ist nicht Teil dieses Belegs. Er wird als grundlegende Steuerungslogik vorausgesetzt und übernimmt die direkte Ansteuerung der mechanischen sowie elektronischen Komponenten.

Der Steuerungscode für das STM32-Board wurde in Python verfasst und ist im Anhang beigefügt. Das Skript beginnt mit dem Import der benötigten Bibliotheken, unter anderem GTK für die grafische Darstellung und OPC UA zur Kommunikation mit der SPS (siehe Codeausschnitt 1). Es öffnet ein grafisches Fenster mit einer Auflösung von 480x272 Pixeln, in dem das Bild des aktuellen Anlagenzustands sowie ein Start-Button geladen werden. Eine der zentralen Funktionen des Programms ist die Verbindung zum OPC UA-Server, die über die IP-Adresse der SPS hergestellt wird (Codeausschnitt 2). Diese Verbindung ermöglicht die kontinuierliche Abfrage des aktuellen Maschinenstatus. Der Zustand der Anlage wird anhand eines deterministischen Zustandsautomaten gesteuert, der über verschiedene Bilder visualisiert wird. Diese Bilder, die den Prozessstatus darstellen, werden durch die Funktion `load_image` skaliert und anschließend im Fenster angezeigt (siehe Codeausschnitt 3). Die Bilder befinden sich in einem Verzeichnis und werden abhängig vom aktuellen Status der Maschine geladen. Dazu fragt das Skript über den OPC UA-Client den Status der SPS ab und aktualisiert das angezeigte Bild alle 500 Millisekunden (Codeausschnitt 4). Zusätzlich zur Visualisierung bietet das Programm eine interaktive Möglichkeit, die Anlage zu starten. Der Start-Button im Fenster ist mit einer Funktion verknüpft, die beim Klicken ein Signal an die SPS sendet, um den Sortivorgang der Anlage zu initiieren (siehe Codeausschnitt 5). Sobald der Button betätigt wird, beginnt die Automatisierungsanlage damit, ein Objekt auf das Förderband zu schieben und entsprechend der Sensordaten zu sortieren. Das Skript bietet außerdem eine benutzerfreundliche Funktionalität zur Umschaltung zwischen Vollbild- und Fenstermodus. Diese wird durch das Drücken der F11-Taste ermöglicht (siehe Codeausschnitt 6). Diese Funktion ist besonders nützlich für die Darstellung der Anlage auf einem kleinen Display, da so der verfügbare Bildschirmplatz optimal genutzt werden kann.

Der Steuerungscode stellt sicher, dass alle Prozesse der Automatisierungsanlage korrekt visualisiert und gesteuert werden. Durch die kontinuierliche Kommunikation mit der SPS können die Prozesse in Echtzeit überwacht werden, was besonders bei industriellen Anwendungen von großer Bedeutung ist.

# 3 Fazit

Im Rahmen dieser Arbeit wurde die Steuerung einer SPS-Automatisierungsanlage durch ein STM32-Board implementiert und getestet. Dabei lag der Fokus auf der Verbindung und dem Zusammenspiel zwischen den Steuerungssystemen und der Automatisierungsanlage. Das Projekt bietet Einblicke in die Möglichkeiten moderner Embedded-Systeme zur Steuerung von Industrieanlagen.

## 3.1 Funktionstest

Der Funktionstest zeigte, dass die Integration des STM32-Boards in die Automatisierungsanlage erfolgreich war. Die Steuerungsaufgaben, insbesondere die Sortierung der metallischen und nicht-metallischen Gegenstände anhand von Sensorwerten, funktionierten wie geplant. Es traten keine unerwarteten Probleme in der Kommunikation zwischen der SPS und dem STM32-Board auf. Das STM32-Boards konnte die Steuerungsaufgaben stabil und zuverlässig abarbeiten und mithilfe des Displays darstellen. Im Github unter <https://github.com/emanuel-eckstein/Embedded-Systems-III> kann das Video **demo.mp4** der laufenden Anlage beobachtet werden.

## 3.2 Ausblick

Zukünftig könnte das System durch zusätzliche Funktionen erweitert werden, etwa durch die Implementierung weiterer Sensoren und Aktoren oder die Optimierung der Steuerung durch Machine-Learning-Algorithmen. Ein weiterer Ansatz wäre eine automatisierte Fehlererkennung, um die Wartung zu vereinfachen.

# Abbildungsverzeichnis

1.1	Automatisierungsanlage, mechanische Bauteile hervorgehoben . . . . .	4
1.2	Automatisierungsanlage, elektrische Bauteile hervorgehoben . . . . .	5
2.1	INIT . . . . .	11
2.2	PRESS_OUT . . . . .	11
2.3	STAMP . . . . .	12
2.4	CHECK . . . . .	12
2.5	STOP . . . . .	12
2.6	nicht-metallisch: FREE . . . . .	13
2.7	metallisch: OUTSOURCE . . . . .	13

# Eidesstattliche Erklärung

Schriftliche Versicherung der selbstständigen Anfertigung

Wir erklären hiermit, dass wir die vorliegende Belegarbeit selbstständig angefertigt, keine anderen als die angegebenen Hilfsmittel benutzt und die Stellen, die im Wortlaut oder im wesentlichen Inhalt aus anderen Werken entnommen wurden, mit genauer Quellenangabe kenntlich gemacht haben.

Leipzig, 11. Oktober 2024



Emanuel Eckstein

Leipzig, 11. Oktober 2024



Tobias Rauch

## Inhaltsverzeichnis (Anhang)

## 1 Python Applikation

Listing 1: Bibliotheken-Import und GTK Setup

```
1 import gi
2
3 gi.require_version("Gtk", "3.0")
4
5 from gi.repository import GLib, Gtk, GdkPixbuf, Gdk
6
7 from opcua import Client, ua
```

Listing 2: Verbindung zu OPC UA-Server herstellen

```
1 # OPC UA Client initialisieren  
2 self.client = Client("opc.tcp://192.168.0.100:4840"  
3 )  
4 self.connect_to_opcua()
```

Listing 3: Bildlade-Funktion

```
1 def load_image(self):
2     # Bild laden
3     image_path = self.image_paths[self.
4         current_image_index]  # Aktuelles Bild
5         verwenden
```

```

4   try:
5       # Urspruengliches Bild laden
6       pixbuf = GdkPixbuf.Pixbuf.new_from_file(
7           image_path)
8
9
10      # Skalierung auf eine Breite von 330 Pixeln
11      new_width = 300
12      new_height = int((new_width / pixbuf.
13                         get_width()) * pixbuf.get_height())
14
15      # Bild skalieren
16      scaled_pixbuf = pixbuf.scale_simple(
17          new_width, new_height, GdkPixbuf.
18          InterpType.BILINEAR)
19      self.image.set_from_pixbuf(scaled_pixbuf)
20  except Exception as e:
21      print(f" Fehler beim Laden des Bildes: {e}")

```

Listing 4: Statusabfrage und Bildaktualisierung

```

1 def update_image(self):
2     # Status abrufen und Bild aktualisieren
3     status = self.status_node.get_value()
4     self.current_image_index = status
5     self.load_image()    # Bild neu laden
6
7     return True    # True zurueckgeben, um die Idle-
8                  # Funktion aufrechtzuerhalten

```

Listing 5: Start-Button Funktionalität

```

1 def on_start_button_clicked(self, widget):
2     # Start-Logik ausfuehren (z.B. OPC UA Node

```

```
    setzen)
3     self.start_node.set_value(ua.DataValue(ua.
4         Variant(True, ua.VariantType.Boolean)))
```

Listing 6: Vollbildmodus Umschalten

```
1 def on_key_press(self, widget, event):
2     # Umschalten des Vollbildmodus bei Druecken von
3     # F11
4     if event.keyval == Gdk.KEY_F11:
5         if self.is_fullscreen:
6             self.unfullscreen()    # Vollbildmodus
7                 verlassen
8         else:
9             self.fullscreen()    # In den
10                Vollbildmodus wechseln
11
12 self.is_fullscreen = not self.is_fullscreen
```