Implementierung eines Tracking Systems für Produkte auf einer Fertigungskette

Studienarbeit T3_3101

Studiengang Embedded Systems

Duale Hochschule Baden-Württemberg Ravensburg, Campus Friedrichshafen

von

Kilian Röper

Abgabedatum:	25. Mai 2025
Bearbeitungszeitraum:	01.10.2024 - 15.07.2025
Matrikelnummer:	5928177
Kurs:	TSA22
Betreuerin / Betreuer:	Markus Wengle

Erklärung

gemäß Ziffer 1.1.14 der Anlage 1 zu §§ 3, 4 und 5 der Studien- und Prüfungsordnung für die Bachelorstudiengänge im Studienbereich Technik der Dualen Hochschule Baden-Württemberg vom 29.09.2017 in der Fassung vom 24.07.2023.

Ich versichere hiermit, dass ich meine Studienarbeit T3_3101 mit dem Thema:

> *Implementierung eines Tracking Systems für Produkte auf einer Fertigungskette*

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.

Friedrichshafen, den 25. Mai 2025

_____

Kilian Röper

Kurzfassung

Die Lernfabrik produziert teilautomatisiert über eine Fertigungskette verschiedene Produkte. Zur Verbesserung der Steuerung, Nachverfolgbarkeit und Dokumentation soll ein System entwickelt werden, das die Position der Produkte mittels NFC-Technologie erfasst und die entsprechenden Daten an ein Manufacturing Execution System (MES) übermittelt. Während geeignete NFC-Tags bereits identifiziert wurden, muss die restliche Systemarchitektur standortspezifisch konzipiert, implementiert und getestet werden.

Ziel dieser Arbeit ist die Entwicklung eines industrietauglichen Systems zur positionsbasierten Produktverfolgung innerhalb der Fertigung. Die erfassten Informationen sollen in Echtzeit an ein vorhandenes System übertragen und dort visualisiert werden.

Zur Entwicklung der Systemarchitektur kamen klassische ingenieurwissenschaftliche Methoden zum Einsatz. Der Fokus lag auf einem kosteneffizienten, störungstoleranten und modular erweiterbaren Aufbau der Hardware- und Softwarekomponenten.

Die Arbeit mündete in einem vollständigen Systementwurf zur produktionsbegleitenden Datenerfassung über NFC. Realisiert wurde ein Prototyp bestehend aus dezentralen Erfassungseinheiten und einer zentralen Steuereinheit, die eine Schnittstelle zum MES bereitstellt. Neben der Hardware wurden die erforderlichen Softwarekomponenten zur Datenübertragung und -visualisierung sowie eine technische Dokumentation für zukünftige Erweiterungen erstellt.

Abstract

The learning factory produces various products semi-automatically via a production chain. To improve control, traceability, and documentation, a system is to be developed that records the position of the products using NFC technology and transmits the corresponding data to a Manufacturing Execution System (MES). While suitable NFC tags have already been identified, the remaining system architecture must be designed, implemented, and tested site-specifically.

The goal of this work is to develop an industrial-grade system for position-based product tracking within production. The recorded information is to be transmitted in real time to an existing system and visualized there.

Classical engineering methods were used to develop the system architecture. The focus was on a cost-effective, fault-tolerant, and modularly expandable design of the hardware and software components.

The work culminated in a complete system design for production-related data acquisition via NFC. A prototype was realized consisting of decentralized acquisition units and a central control unit that provides an interface to the MES. In addition to the hardware, the necessary software components for data transmission and visualization as well as technical documentation for future extensions were created.

Inhaltsverzeichnis

1 Einleitung

Mit der zunehmenden Automatisierung industrieller Produktionsprozesse wächst der Bedarf an intelligenten Systemen, die eine lückenlose Nachverfolgbarkeit und effiziente Steuerung ermöglichen. Besonders in vernetzten Fertigungsumgebungen, wie sie im Kontext von Industrie-4.0 realisiert werden, spielt die automatische Identifikation von Produkten entlang der Fertigungskette eine zentrale Rolle. Zum Einsatz kommen dafür verschiedene Technologien - von optischer Erkennung bis hin zu RFID und NFC (Near Field Communication).

Die Lernfabrik der DHBW bietet eine reale, teilautomatisierte Fertigungsumgebung, in der solche Konzepte unter praxisnahen Bedingungen erprobt werden können. Aktuell fehlt jedoch ein System zur durchgängigen Identifikation und Nachverfolgung von Produkten innerhalb der Fertigungslinie. Dies erschwert nicht nur die Prozessüberwachung, sondern limitiert auch Möglichkeiten zur Datenanalyse, Optimierung und Dokumentation.

Ziel dieser Arbeit ist die Entwicklung eines Systems, das die Position und Identität von Produkten während des Fertigungsprozesses mit Hilfe von NFC-Sensoren automatisch erfasst. Zusätzlich soll es möglich sein Informationen automatisch auf die NFC-Tag zu schreiben. Die erfassten Daten werden an ein bestehendes Manufacturing Execution System (MES) übermittelt und dort zur Visualisierung bereitgestellt. Darüber hinaus sollen die von den NFC-Sensoren erfassten Informationen genutzt werden, um den logischen Ablauf des Bandumlaufsystems zu steuern. Hierfür ist ein Kommunikationsmechanismus zwischen den einzelnen Bandumlaufstationen und den NFC-Sensoren zu entwerfen. Im Rahmen dieser Arbeit wird eine geeignete Systemarchitektur für die Hard- und Softwarekomponenten konzipiert, implementiert und evaluiert.

Die zentrale Fragestellung der Arbeit lautet:

> *Wie kann ein robustes, kosteneffizientes und erweiterbares Trackingsystem für Produkte in einer bestehenden Fertigungsumgebung mithilfe von NFC-Technologie gestaltet und in ein MES integriert werden?*

Zur Beantwortung dieser Frage werden klassische ingenieurwissenschaftliche Methoden wie Anforderungsanalyse, Systementwurf, prototypische Umsetzung und Validierung angewendet. Neben der technischen Umsetzung liegt ein besonderes Augenmerk auf der industriellen Tauglichkeit sowie der zukünftigen Erweiterbarkeit des Systems.

Die Arbeit gliedert sich wie folgt: Kapitel 2 beschreibt die theoretischen und technischen Grundlagen der verwendeten Technologien. Kapitel 3 analysiert die Anforderungen an das System und erläutert das methodische Vorgehen. Kapitel 4 enthält den Entwurf der Systemarchitektur sowie die Darstellung und Auswertung der Ergebnisse. Abschließend fasst Kapitel 5 die wesentlichen Erkenntnisse zusammen und gibt einen Ausblick auf mögliche Weiterentwicklungen.

2 Grundlagen

Dieses Kapitel stellt die theoretischen und technischen Grundlagen bereit, die für das Verständnis und die Umsetzung der im weiteren Verlauf beschriebenen Konzepte und Lösungen erforderlich sind. Dazu gehören sowohl technologische Aspekte – wie die Funktionsweise von RFID/NFC-Systemen oder des MQTT-Protokolls – als auch methodische Grundlagen im Bereich der Kommunikation und Steuerung verteilter Systeme.

Ziel ist es, ein solides Fundament zu schaffen, auf dem die spätere Systemkonzeption und Implementierung aufbauen kann. Die hier dargestellten Inhalte stehen daher in engem Zusammenhang mit den folgenden Kapiteln und werden dort zur Anwendung gebracht – beispielsweise bei der Auswahl geeigneter Kommunikationsprotokolle, Hardwarekomponenten oder der Integration in die Gesamtarchitektur des Systems.

2.1 RFID und NFC-Technologie

Near Field Communication (NFC) ist eine auf der RFID-Technologie (radio-frequency identification) basierende drahtlose Kommunikationstechnologie, die speziell für die Kommunikation über kurze Distanzen von wenigen Zentimetern entwickelt wurde. Sie ermöglicht den kontaktlosen Datenaustausch zwischen zwei NFC-fähigen Geräten, wobei ein Gerät aktiv (z.B. ein Smartphone) und das andere passiv (z.B. ein NFC-Tag) sein kann. Wesentliche Unterschiede zwischen RFID und NFC sind in Tabelle 2.1 festgehalten.

Tabelle 2.1: Vergleich zwischen RFID und NFC

Eigenschaft	**RFID**	**NFC**
Frequenzbereich	Verschiedene Bereiche: LF (125–134 kHz), HF (13,56 MHz), UHF (860–960 MHz)	HF (13,56 MHz)
Kommunikations-Richtung	In der Regel unidirektional (Reader zu Tag)	Bidirektional (Peer-to-Peer möglich)
Reichweite	Bis zu mehreren Metern (je nach Frequenz)	Typisch 0–10 cm
Datenrate	Abhängig vom Standard, z.B. bis zu 640 kbit/s (UHF)	Bis zu 424 kbit/s
Typische Anwendungen	Lagerlogistik, Zutrittskontrolle, Tierkennzeichnung, Supply Chain	Mobile Payment, Geräteverbindung, Zugangskontrolle im Konsumerbereich
Standardisierung	ISO/IEC 18000, ISO/IEC 14443, 15693 usw.	ISO/IEC 14443, ISO/IEC 18092, NFC Forum Spezifikationen
Interoperabilität	Eingeschränkt (herstellerspezifisch, viele Standards)	Hoch (interoperabel durch NFC Forum Vorgaben)

NFC arbeitet in einem Frequenzbereich von 13,56MHz und ermöglicht Datenraten von bis zu 424kbit/s. Der wichtigste Unterschied zu klassischen RFID-Systemen liegt in der bidirektionalen Kommunikation: Während RFID meist eine unidirektionale Leser-Tag-Kommunikation darstellt, ermöglicht NFC einen wechselseitigen Datenaustausch. Die typische maximale Reichweite beträgt etwa 10cm, wodurch eine gezielte, sichere Interaktion gewährleistet wird [Fin23, Seite 47 Tabelle 3.6].

Die Technologie kommt u.a. bei mobiler Bezahlung (z.B. Apple Pay), elektronischen Zugangssystemen, öffentlichen Verkehrsmitteln sowie bei der schnellen Geräteverbindung (z.B. Bluetooth-Kopplung) zum Einsatz. In dieser Arbeit wird die Technologie zum Austausch von Informationen zwischen NFC-Readern und Tags an einer Fertigungskette verwendet. In Kapitel 3 wird genauer darauf eingegangen wie hierfür die Integration in die Gesamtarchitektur erfolgt. Zur Implementierung von NFC-basierten

Funktionen stehen für Embedded-Systeme verschiedene Softwarebibliotheken zur Verfügung (vgl. 3.3). Diese ermöglichen etwa das Auslesen und Schreiben von NFC-Tags oder das Initiieren von Peer-to-Peer-Verbindungen.

2.2 Das MQTT-Protokoll

MQTT (Message Queuing Telemetry Transport) ist ein Client-Server-Publish/Subscribe-Nachrichtentransportprotokoll. Es ist leichtgewichtig, offen, einfach und leicht zu implementieren. Diese Eigenschaften prädestinieren es für den Einsatz in vielen Situationen, auch in eingeschränkten Umgebungen, wie z. B. für die Kommunikation im Machine-to-Machine- (M2M) und Internet of Things-Kontext (IoT), wo ein geringer Code-Footprint erforderlich ist und/oder die Netzwerkbandbreite knapp ist [OAS19, Abstract].

Das Protokoll läuft über TCP/IP oder andere Netzwerkprotokolle, die geordnete, verlustfreie und bidirektionale Verbindungen ermöglichen. Zu seinen Funktionen gehören:

- Verwendung des Publish/Subscribe-Nachrichtenmusters, das eine One-to-Many-Nachrichtenverteilung und die Entkopplung von Anwendungen ermöglicht.
- Drei Dienstqualitäten für die Nachrichtenübermittlung
- Ein Nachrichtentransport, der vom Inhalt der Nutzlast unabhängig ist.
- Geringer Transportaufwand und minimierter Protokollaustausch reduzieren den Netzwerkverkehr.
- Ein Mechanismus zur Benachrichtigung interessierter Parteien bei einer ungewöhnlichen Verbindungsunterbrechung.

Das Protokoll wurde mit dem Ziel entwickelt, eine möglichst leichtgewichtige und ressourcenschonende Kommunikation zu ermöglichen, insbesondere über Netzwerke mit hoher Latenz oder geringer Bandbreite. Im Vergleich zu traditionellen Client-Server-Modellen verwendet MQTT das Publish-/Subscribe-Prinzip, das eine entkoppelte Kommunikation erlaubt. In dieser Arbeit wird MQTT als Kommunikationsschicht zwischen Sensoren und einem Raspberry Pi eingesetzt [3.3].

Im Folgenden werden die zentralen Konzepte des Protokolls näher erläutert: das Publish-/Subscribe-Prinzip, die Rollen von Client und Broker sowie die verschiedenen Quality-of-Service-Stufen.

Publish-/Subscribe-Prinzip

MQTT basiert auf einem Publish-/Subscribe-Modell. Ein *Publisher* sendet Nachrichten an ein bestimmtes *Topic*, ohne dabei explizit anzugeben, wer diese Nachrichten empfangen soll. Topics dienen in MQTT als hierarchisches Adressierungsschema zur Organisation von Nachrichten. Sie sind UTF-8-kodierte Strings, die durch Schrägstriche (/") strukturiert sind, z.B. `sensor/temperatur/wohnzimmer`. Ein *Subscriber* hingegen abonniert ein oder mehrere Topics und erhält automatisch alle Nachrichten, die zu diesen Topics veröffentlicht werden [Abbildung 2.1].

Client und Broker

Die zentrale Vermittlungsinstanz in diesem Modell ist der sogenannte *Broker*. Er empfängt alle vom Publisher gesendeten Nachrichten und leitet sie an die entsprechenden Subscriber weiter. Jeder Teilnehmer in einem MQTT-Netzwerk ist ein *Client*, unabhängig davon, ob er Nachrichten veröffentlicht oder abonniert. Der Broker übernimmt die vollständige Verwaltung der Kommunikation: Er sorgt dafür, dass Nachrichten zuverlässig und effizient verteilt werden, ohne dass sich Publisher und Subscriber gegenseitig kennen oder direkt kommunizieren müssen. Diese Architektur entkoppelt

Sender und Empfänger sowohl räumlich als auch zeitlich [OAS19, Abschnitt 4.7] [Abbildung 2.1].

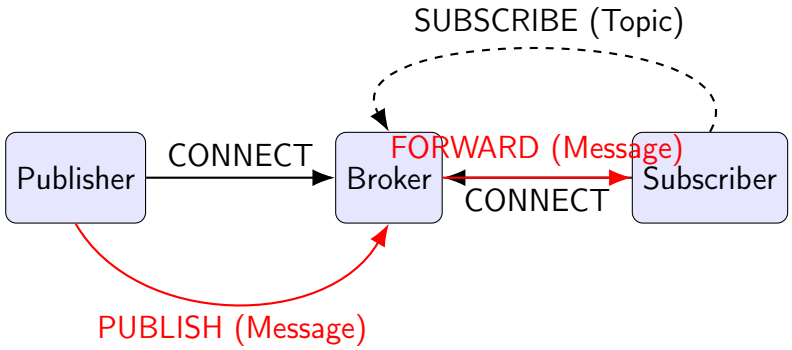

Abbildung 2.1: Ablauf des MQTT Publish-/Subscribe-Verfahrens mit Verbindungsaufbau

Quality of Service (QoS)

MQTT bietet drei verschiedene Stufen der Nachrichten-Zustellungssicherheit (Quality of Service), die je nach Anwendungsfall gewählt werden können:

- **QoS 0** – ”höchstens einmal”: Nachrichten werden nach bestem Wissen und Gewissen der Betriebsumgebung zugestellt. Nachrichtenverlust kann auftreten. Diese Stufe kann beispielsweise bei Umgebungssensordaten eingesetzt werden, bei denen der Verlust eines einzelnen Messwerts unerheblich ist, da der nächste kurz darauf veröffentlicht wird.

- **QoS 1** – ”mindestens einmal”: Nachrichten werden garantiert ankommen, es können aber Duplikate auftreten.

- **QoS 2** – ”genau einmal”: Nachrichten werden garantiert genau einmal ankommen. Diese Stufe kann beispielsweise bei Abrechnungssystemen eingesetzt werden, bei denen doppelte oder verlorene Nachrichten zu falschen Gebühren führen können. [OAS19, Abschnitt 4.3].

2.3 Node-Red - Eine Steuerungs- und Visualisierungssoftware

Node-RED ist ein von IBM entwickeltes Open-Source-Tool zur visuellen Programmierung, das auf Node.js basiert. Es ermöglicht die Erstellung von Anwendungen durch das Verbinden sogenannter Nodes in einem webbasierten Editor. Ursprünglich für das Internet der Dinge (IoT) konzipiert, findet Node-RED heute Anwendung in Bereichen wie Heimautomation, industrieller Steuerung und Datenintegration [Nod24].

Die zentrale Einheit in Node-RED ist der Flow, eine Abfolge von miteinander verbundenen Nodes, die Daten verarbeiten und weiterleiten. Jede Node erfüllt dabei eine spezifische Funktion, z.B. das Empfangen von Daten, deren Verarbeitung oder das Senden von Ergebnissen. Die Kommunikation zwischen den Nodes erfolgt über standardisierte Nachrichtenobjekte im JSON-Format, die typischerweise folgende Eigenschaften enthalten:

- msgid: Eine eindeutige Kennung der Nachricht
- payload: Die eigentlichen Nutzdaten
- topic: Ein optionaler Themenbezeichner zur Kategorisierung

Diese Struktur bildet die Grundlage für die Datenverarbeitung in Node-RED. Ein beispielhafter Flow ist in Abbildung 2.2 abgebildet.

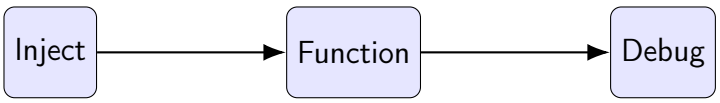

Abbildung 2.2: Ein einfacher Node-RED-Flow: Inject → Function → Debug

Ein *Inject*-Node erzeugt eine Nachricht mit einem definierten *Payload*. Ein *Function*-Node verarbeitet die Nachricht, beispielsweise durch Anwendung einer JavaScript-Funktion. Der *Debug*-Node gibt das Ergebnis im Debug-Fenster des Editors aus. Sol-

che Flows ermöglichen eine schnelle und intuitive Entwicklung von Anwendungen, ohne tiefgreifende Programmierkenntnisse.

Darüber hinaus bietet Node-RED die Möglichkeit, grafische Benutzeroberflächen mithilfe des sogenannten *Node-RED Dashboards* zu erstellen. Diese Oberflächen können in einem separaten Browser-Fenster aufgerufen werden und eignen sich insbesondere zur Echtzeitvisualisierung von Prozessdaten. In Kapitel 4 wird diese Funktionalität exemplarisch genutzt, um eine interaktive Darstellung des Bandumlaufsystems zu realisieren und Benutzereingaben zur Steuerung zu ermöglichen.

2.4 Speicherprogrammierbare Steuerungen (SPS)

Speicherprogrammierbare Steuerungen (SPS) sind spezialisierte, industrielle Recheneinheiten zur Automatisierung technischer Prozesse. Sie ersetzen konventionelle, fest verdrahtete Steuerungssysteme und bieten eine flexible, softwarebasierte Möglichkeit zur Realisierung komplexer Ablaufsteuerungen. SPS-Systeme sind in nahezu allen Bereichen der Industrieautomation zu finden – von einfachen Förderbändern bis hin zu hochkomplexen Produktionslinien [Sie21].

2.4.1 Aufbau einer SPS

Eine SPS besteht aus mehreren funktionalen Einheiten: einer zentralen Recheneinheit (CPU), digitalen oder analogen Ein- und Ausgabemodulen (I/O), einer Spannungsversorgung sowie optionalen Kommunikationsschnittstellen und Programmierschnittstellen. Das folgende Diagramm zeigt den grundsätzlichen Aufbau:

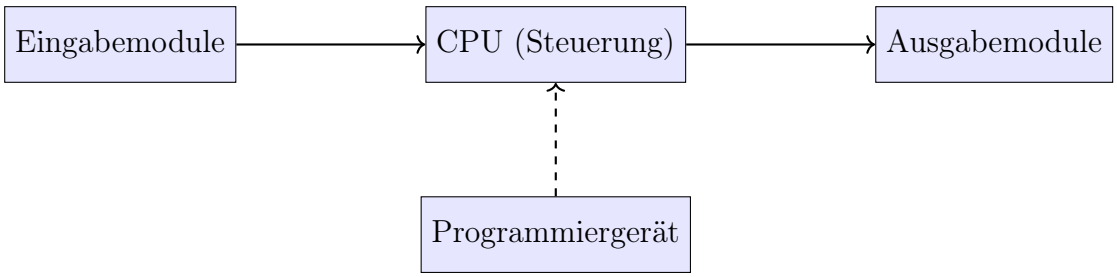

Abbildung 2.3: Schematischer Aufbau einer SPS

2.4.2 Funktionsweise: Der SPS-Zyklus

Die Arbeitsweise einer SPS basiert auf dem EVA-Prinzip (Eingabe – Verarbeitung – Ausgabe). Dabei werden zyklisch alle Eingänge gelesen, der Steuerungsalgorithmus (z.B. in Form eines SPS-Programms) verarbeitet und anschließend die entsprechenden Ausgänge gesetzt. Dieser Ablauf, auch *SPS-Zyklus* genannt, wiederholt sich kontinuierlich.

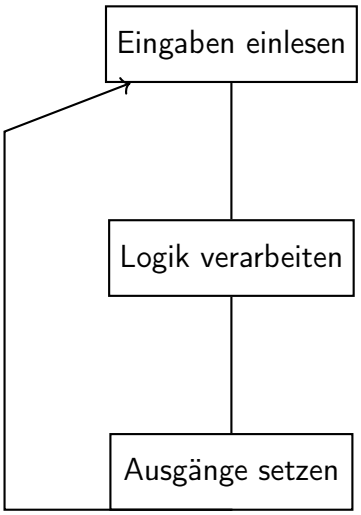

Abbildung 2.4: Zyklus einer SPS-Steuerung

2.4.3 Programmiersprachen nach IEC 61131-3

Die Norm IEC 61131-3 definiert fünf standardisierte Programmiersprachen für SPS:

- **KOP (Kontaktplan)** – grafische Darstellung mit Relaislogik
- **FUP (Funktionsplan)** – logische Gatterstruktur (ähnlich wie digitale Schaltungen)
- **AWL (Anweisungsliste)** – Assembler-ähnlich
- **ST (Structured Text)** – textbasierte Hochsprache (ähnlich Pascal)
- **SFC (Sequential Function Chart)** – Ablaufsteuerungen mit Schritten und Transitionen

Ein Beispiel in *Structured Text (ST)* zeigt eine einfache UND-Verknüpfung zweier Eingänge:

Listing 2.1: Beispielprogramm in Structured Text

```
1 IF Eingang1 = TRUE AND Eingang2 = FALSE THEN
2 Ausgang := TRUE;
3 ELSE
4 Ausgang := FALSE;
5 END_IF;
```

2.5 Serial Peripheral Protocoll (SPI)

3 Konzeptentwurf

Dieses Kapitel beschreibt die methodische Herleitung eines technischen Konzepts zur Umsetzung des in Kapitel 2 vorgestellten Systems. Basierend auf den identifizierten technologischen Grundlagen erfolgt zunächst eine strukturierte Analyse der funktionalen und nicht-funktionalen Anforderungen mithilfe des Requirements Engineerings.

Auf dieser Grundlage werden mehrere Lösungsoptionen konzipiert, die für den Einsatz unterschiedlicher Hardwarekomponenten geeignet sind. Ein wesentlicher Bestandteil des Konzepts ist der Systementwurf. In diesem wird die Architektur der miteineinander interagierenden Hardwarekomponenten beschrieben. Außerdem wird erklärt mit welchen Software-Bibliotheken gearbeitet wurde und wie die Informationen im Speicher der NFC-Tags organisiert ist.

Ziel ist es, eine nachvollziehbare und belastbare Begründung für die gewählte Vorgehensweise zu liefern, die die Anforderungen der Anwendung erfüllt und eine verlässliche Grundlage für die anschließende Umsetzung bietet.

3.1 Anforderungsanalyse mittels Requirements Engineering

Im Rahmen dieser Arbeit bildet das Requirements Engineering die methodische Basis für die Konzeption eines funktionalen und technisch realisierbaren Systems. Dabei werden sowohl funktionale als auch nicht-funktionale Anforderungen betrachtet. Die

gewonnenen Erkenntnisse dienen anschließend als Ausgangspunkt für den Systementwurf und die Auswahl geeigneter technischer Komponenten.

3.1.1 Funktionale Anforderungen

Um die Anforderungen an das zu entwickelnde System bestimmen zu können, soll zunächst die Funktionalität aus der Sicht des Anwenders in Anforderungen festgehalten werden.

Tabelle 3.1: funktionale Systemanforderungen

ID	**Anforderung**
F01	Der Anwender soll Produkte auf einer Fertigungskette mittels einer eindeutigen und einmaligen Identifikationsnummer identifizieren können.
F02	Der Anwender soll über die Position und den Zustand der Produkte auf dem Bandumlaufsystem informiert werden können.
F03	Die Positionen und Zustände der Produkte sollen visuell dargestellt werden.
F04	Das Bandumlaufsystem soll basierend auf den Produktzuständen gesteuert werden können.
F05	Die Produkte sollen einen NFC-Tag mit allen relevanten Zustandsinformationen und der eindeutigen Identifikationsnummer tragen.
F06	Über vier NFC-Lesegeräte sollen die Position und die Zustände der Produkte ermittelt und geschrieben werden können.
F07	Die NFC-Lesegeräte müssen ein Verweilen eines Produktes an einer Station erkennen können.
F08	Das Tracking-System soll den letzten Zustand eines Produktes speichern.
F09	Die Speicherung von Zuständen soll auf den NFC-Tags und im System erfolgen.

3.1.2 Nicht funktionale Anforderungen

Nachdem die Funktion des Systems ermittelt wurde, sollen entsprechende technische, nicht funktionale Anforderungen festgehalten werden.

Tabelle 3.2: nicht-funktionale Systemanforderungen

ID	**Anforderung**
NF01	Die Kommunikation von Informationen von und zu den NFC-Lesegeräten soll über ein leichtgewichtiges Netzwerkprotokoll erfolgen.
NF02	Die Netzwerkkommunikation sollte drahtlos funktionieren.
NF03	MQTT-Nachrichten können mit einer Quality-of-Service-Stufe 0 gesendet werden.
NF04	Die Software der Mikrocontroller soll generisch aufgebaut sein und logische Erweiterungen ermöglichen.
NF05	Diskrepanzen von Zuständen im System und auf den Tags dürfen nicht entstehen. Eine Behandlung des Problems muss in Software abgebildet werden.
NF06	Das Tracking-System soll über eine Anbindung zu einer SPS verfügen.
NF07	Das Bandumlaufsystem soll weiterhin über die bestehende SPS gesteuert werden.
NF08	Node-Red soll als Visualisierungssoftware verwendet werden.

3.2 Hardwarebewertung mithilfe der Nutzwertanalyse

Die zur Auswahl nötige Bewertung der Hardware soll basierend auf einer Nutzwertanalyse passieren. Drei verschiedene Hardwareklassen sollen betrachtet werden, um den Systemebenen Hardwarekomponenten zuweisen zu können. Dazu gehören die NFC-Tags, die NFC-Lesegeräte und die Controller-Einheiten.

3.2.1 NFC-Tags

NFC-Tags unterscheiden sich im Hinblick auf unterschiedliche Faktoren. Dazu zählen beispielsweise: Speicherkapazität, Form und Größe. Das NFC Forum definiert fünf Tag-Typen (Typ 1 bis Typ 5), die auf existierenden RFID-Technologien basieren [Tabelle 3.3].

Tabelle 3.3: Vergleich der NFC-Tag-Typen gemäß NFC Forum [NFC24]

Tag-Typ	**Technologie-Basis**	**Speicher**	**Datenrate**	**Sicherheit**	**Besonderheiten**
Typ 1	Innovision Topaz	96B – 2KB	106kbit/s	Keine Authentifizierung	Kostengünstig, einfache Anwendungen
Typ 2	NXP MIFARE Ultralight und NTAG	bis 2KB	106kbit/s	Einfach	Häufig in Tickets, Werbeanwendungen
Typ 3	Sony FeliCa	bis 1MB	212–424 kbit/s	Zugriffskontrolle möglich	Hohe Geschwindigkeit, verbreitet in Japan
Typ 4	NXP DESFire, Smartcard	bis 32KB+	106–424 kbit/s	Hohe Sicherheit (AES, ISO 7816)	Zutrittssysteme, sichere Speicherung
Typ 5	ST25, NXP ICODE (ISO 15693)	bis ca. 64KB	bis 53kbit/s	Variabel	Größere Reichweite, Industrieeinsatz

Basierend auf gegebenen Informationen zu den Tag-Typen wurde die Nutzwertanalyse durchgeführt. Die Eigenschaften wurden entsprechend als Bewertungskriterien gewichtet, wodurch nach zusammenzählen der vergebenen Punkte der passende Tag-Typ und aus 3.3 das passende Produkt ermittelt werden konnte.

Die höchste Gewichtung hat in der Analyse die Lesegeschwindigkeit erhalten, da am Bandumlaufsystem eine Verarbeitung von Informationen in Echtzeit relevant ist. Das folgt daraus, dass der Zustand eines Produktes an einer Station ermittelt und dann auf den Tag geschrieben werden muss [3.4].

Tabelle 3.4: Nutzwertanalyse zur Bewertung von NFC-Tag-Typen (1 - 5, wobei 5: sehr gut

Kriterium	**Gewichtung**	**Typ 1**	**Typ 2**	**Typ 3**	**Typ 4**	**Typ 5**
Lesegeschwindigkeit	25%	2	3	2	4	5
Speicherkapazität	20%	2	3	4	5	4
Kosten	20%	5	4	3	2	3
Kompatibilität	15%	3	4	2	5	4
Energieeffizienz (passiv)	10%	5	5	4	3	4
Verfügbarkeit am Markt	10%	3	5	2	4	4
Gesamtnutzwert		**3.10**	**4.00**	**2.85**	**4.00**	**4.10**

Tag Typ 2 ist nach dieser Analyse der favorisierte Typ. Für die Arbeit wurde entsprechend ein NXP NTAG213-Tag gewählt. Dieser bietet einen EEPROM-Speicher von 144 Bytes, der in 4 Byte große Seiten (Pages) unterteilt ist. Daraus ergeben sich 36 adressierbare Blöcke. Von diesen sind laut Datenblatt etwa 112 Bytes frei vom Anwender beschreibbar [NXP14], da einige Speicherbereiche für Systemfunktionen wie die UID, Konfiguration und Sicherheitsmechanismen reserviert sind. Das genügt für das Schreiben von binärcodierten Zuständen und Tracking-Positionen auf den Tag. Entsprechend wurde eine Codierungstabelle in Abschnitt 3.3 ausgearbeitet.

3.2.2 NFC-Lesegeräte

Da nach der Auswahl des NFC-Tags eine entsprechende Kompatibilität gewährleistet sein muss, reduziert sich die Vielfalt an geeigneten Lesegeräten. Aus diesem Grund wurde auf eine detaillierte Nutzwertanalyse verzichtet, da es sich nicht um gleichwertig vergleichbare Alternativen handelt, sondern primär um die Auswahl eines technisch kompatiblen und verfügbaren Bauteils.

Insbesondere muss das NFC-Lesegerät den Frequenzbereich von 13,56 MHz unterstützen und mit ISO/IEC 14443 Typ A kompatibel sein, um mit dem gewählten NTAG213-Tag kommunizieren zu können. Zusätzlich waren Kriterien wie Verfügbarkeit, Preis und Schnittstellenunterstützung entscheidend.

Der PN532 von NXP hat sich als geeignete Wahl erwiesen. Das Modul unterstützt sowohl I^2C als auch SPI als Kommunikationsschnittstelle und bietet eine effektive Reichweite von ca. 5–10 cm. Diese ist für das vorliegende System ausreichend, da die Produkte gezielt an den Sensorstationen positioniert werden und dort die Datenübertragung stattfindet.

Als bevorzugte Schnittstelle zwischen Mikrocontroller und NFC-Modul wurde SPI gewählt, da sie gegenüber I^2C eine höhere Datenrate, robustere Signalübertragung und eine bessere Unterstützung bei Echtzeitanforderungen bietet.

3.2.3 Controller-Einheiten

Tabelle 3.5: Nutzwertanalyse zur Bewertung der Mikrocontroller(1 - 5, wobei 5: sehr gut)

Kriterium	**Gewichtung**	**ESP8266**	**ESP32**
Rechenleistung	20%	3	5
Energieverbrauch	25%	4	3
GPIO-Pins	15%	3	5
kabellos netzwerkfähig	20%	5	5
Preis	20%	5	4
Gesamtnutzwert		**4,05**	**4,3**

Die Nutzwertanalyse wurde auf zwei Mikrocontroller eingegrenzt: den ESP8266 und den ESP32. Beide Module bieten eine solide Grundlage zur Implementierung von Anwendungen mit geringer bis mittlerer Komplexität, verfügen über integrierte Netzwerkschnittstellen, sind breit verfügbar und gut dokumentiert (vgl. Tabelle 3.5). Aufgrund dieser Gemeinsamkeiten wurde auf die Einbeziehung weiterer Hardwarevarianten verzichtet.

Das Ergebnis der Analyse zeigt, dass der ESP32 insgesamt eine höhere Nutzwertpunktzahl erreicht. Dennoch fiel die Entscheidung in diesem Projekt zugunsten des ESP8266 aus, da einige der Stärken des ESP32 für den vorliegenden Anwendungsfall nicht ausschlaggebend sind.

So verfügt der ESP32 über zwei CPU-Kerne, wodurch parallele Prozesse (Multithreading) möglich werden. Dies kann bei gleichzeitiger Verarbeitung von Kommunikations- und Steuerungsaufgaben vorteilhaft sein. Im betrachteten System ist jedoch eine sequenzielle Verarbeitung – etwa durch Interrupt-gesteuerte Abläufe – ausreichend, da beispielsweise das Schreiben auf einen NFC-Tag erst nach vollständigem Empfang der Nachricht über MQTT erfolgt. Somit ergibt sich hier kein zwingender Vorteil für die Mehrkernarchitektur.

Ein weiterer Unterschied besteht in der Anzahl der verfügbaren GPIO-Pins: Der ESP32 bietet mehr konfigurierbare Pins, was insbesondere bei komplexeren Schaltungen hilfreich sein kann. Beim ESP8266 hingegen kam es im Testbetrieb vereinzelt zu Einschränkungen aufgrund mehrfach belegter Pins. Beispielsweise musste das Slave-Select-Signal des SPI-Busses auf einen alternativen Pin gelegt werden, da die ursprüngliche Konfiguration den Flash-Vorgang blockierte. Dennoch stehen auch beim ESP8266 ausreichend Pins zur Verfügung, um die Anforderungen dieses Projekts zuverlässig zu erfüllen.

Hinsichtlich Energieverbrauch und Preis schneidet der ESP8266 besser ab. Diese Vorteile sind auf den geringeren Systemumfang und die reduzierte Hardwarekomplexität zurückzuführen – ein relevantes Kriterium für ressourcenschonende Embedded-Anwendungen. Insgesamt erfüllt der ESP8266 damit alle funktionalen Anforderungen des Systems und stellt eine technisch sinnvolle Wahl dar, um mit dem NFC-Lesegerät zu kommunizieren. Der ESP32 wäre jedoch ebenfalls eine geeignete Alternative mit erweiterter Funktionalität für zukünftige Skalierungen.

3.2.4 Zentraleinheit: Raspberry Pi

Für die zentrale Verarbeitung und Weiterleitung der MQTT-Nachrichten wurde ein Raspberry Pi 3B+ als Systemzentrale eingesetzt. Dieses Gerät erfüllt die Anforderungen an eine kontinuierliche Betriebsbereitschaft, stabile Netzwerkkommunikation und eine ausreichende Rechenleistung für die Verwaltung eines erhöhten Nachrichtendurchsatzes.

Eine Einbeziehung des Raspberry Pi in die zuvor dargestellte Nutzwertanalyse der Mikrocontroller erfolgte bewusst nicht, da er auf einer übergeordneten Systemebene agiert. Während die Mikrocontroller-Einheiten an den Stationen in direktem Kontakt mit den Sensoren stehen und deren Steuerung übernehmen, fungiert der Raspberry Pi als dedizierte Zentraleinheit zur Aggregation und Verteilung der Systemdaten über das MQTT-Protokoll.

Aufgrund der ausgezeichneten Softwareunterstützung, der nativen Kompatibilität mit MQTT-Broker-Implementierungen (z. B. Mosquitto) sowie der Möglichkeit zur einfachen Netzwerk- und Systemintegration wurde auf eine vergleichende Analyse alternativer Hardwareplattformen verzichtet. Der Raspberry Pi stellte hier eine erprobte und funktional bewährte Lösung dar.

3.3 Systementwurf

3.3.1 Organisation der Hardwarekomponenten

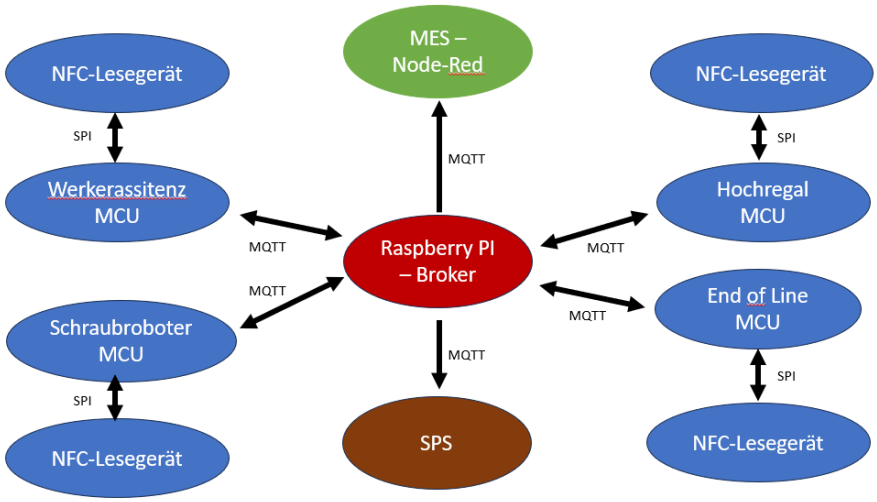

Abbildung 3.1: Systemarchitektur des NFC-Tracking-Systems (MCU: Microcontroller Unit –> ESP8266)

Die Hardwarekomponenten des Systems sind entsprechend der in Abbildung 3.1 dargestellten Architektur organisiert. Ab den Mikrocontrollern erfolgt der Informationsaustausch über das MQTT-Protokoll (vgl. Abschnitt Abschnitt 2.2). Die ESP8266-Mikrocontroller sind über das SPI-Protokoll (Serial Peripheral Interface, vgl. Abschnitt 2.5) mit den jeweiligen NFC-Lesegeräten (vgl. Abschnitt 2.1) verbunden und kommunizieren anschließend drahtlos mit dem zentralen Raspberry Pi.

Der Raspberry Pi fungiert als zentrale Einheit und ist ebenfalls drahtlos mit der speicherprogrammierbaren Steuerung (SPS, vgl. Abschnitt 2.4) sowie mit dem MES-System (Manufacturing Execution System, vgl. Abschnitt 2.3) verbunden. Das MES-System setzt sich in dieser Projektbeschreibung ausschließlich aus dem Node-Red-Dashboard zusammen, beinhaltet aber noch weitere Systemkomponenten.

3.3.2 Speicherorganisation der NFC-Tags

Tabelle 3.6: Mögliche Zustände im Tracking-Block

Hex-Wert	**Zustand**
0x01	to_shelf
0x02	at_shelf
0x03	in_shelf
0x04	to_screwing
0x05	at_screwing
0x06	to_eol
0x07	at_eol
0x08	at_assembling
0x09	to_assembling

Tabelle 3.7: Zuordnung der Speicherbereiche auf dem NFC-Tag

Datenfeld	**Speicherblock (Hex-Adresse)**
UID	0x0A
assembling	0x0B
shelf	0x0C
screwing	0x0D
end_of_line	0x0E
tracking	0x0F

Die Zustände der verschiedenen Stationen werden auf dem NFC-Tag in vordefinierte Speicherblöcke geschrieben (siehe Tabelle 3.7). Zur Speicherung eines neuen Zustands, beispielsweise des Montagezustands, wird die folgende Methode verwendet:

Listing 3.1: Speicherung eines Zustands im Speicherblock `ASSEMBLING_BLOCK`

```
1 success = nfc.mifareultralight_WritePage(ASSEMBLING_BLOCK,
    io_state_temp);
```

Dabei bezeichnet `ASSEMBLING_BLOCK` die Adresse des zu beschreibenden Speicherbereichs (Block `0x0B`), und `io_state_temp` enthält den zu speichernden Wert, z. B. den aktuellen Zustand als 4-Byte-Wort. Die Funktion `mifareultralight_WritePage()` gibt an, ob der Schreibvorgang erfolgreich war.

Diese Methode wird für jede Station analog verwendet, wobei jeweils der entsprechende Block gewählt wird.

4 Umsetzung der Softwarearchitektur

4.1 Softwaredesign auf den Mikrocontrollern

Softwarearchitektur und Designtools

Die Softwarearchitektur auf den eingesetzten ESP8266-Mikrocontrollern wurde mit der Arduino-IDE angefertigt. Sie gliedert sich in zwei Hauptkomponenten:

- **Setup-Funktion:** Verantwortlich für die Überprüfung der Kommunikationsfähigkeit der erstellten Kommunikations-Objekte.

- **Loop-Funktion:** Verwaltet den Zugriff auf den MQTT-Client, den PN532-Leser sowie das Auslesen und Schreiben der Speicherblöcke.

Systemstart und Initialisierung

Beim Systemstart werden zunächst globale Variablen und Objektinstanzen initialisiert, die für die spätere Kommunikation und Datenverarbeitung erforderlich sind [siehe Listing 4.1]. Dazu zählen:

- Ein WiFi-Objekt, das die Netzwerkverbindung herstellt und damit die Grundlage für die MQTT-Kommunikation bildet.

- Ein SPI-Objekt, das die serielle Verbindung zum PN532-NFC-Sensor über das SPI-Protokoll ermöglicht.

Basierend auf diesen Kommunikationsschnittstellen werden zwei zentrale Objekte global angelegt:

- Ein NFC-Objekt, das über das SPI-Objekt mit dem PN532-Sensor kommuniziert.

- Ein MQTT-Objekt, das über das WiFi-Objekt mit dem MQTT-Broker verbunden wird.

Zusätzlich werden globale Variablen definiert, die während der Laufzeit von verschiedenen Programmteilen gelesen und verändert werden können. Dazu gehören unter anderem:

- die UID-Variable, welche die eindeutige Kennung des erfassten NFC-Tags speichert,

- sowie die io_state-Variable, die den aktuellen Verarbeitungszustand an der Station eines Produkts repräsentiert.

Listing 4.1: Initialisierung globaler Objekte und Variablen

```
1  // Globale Kommunikationsobjekte
2  // SPI-connection
3  PN532_SPI intf(SPI, PN532_SS);
4  PN532 nfc = PN532(intf);
5  // wifi and mqtt client
6  WiFiClient wifiClient;
7  MqttClient mqttClient(wifiClient);
8
9  // Globale Zustandsvariablen
10 uint8_t uid_data[UID_LENGTH] = {0};
11 uint8_t io_state[4] = {0};
```

In der Setup-Funktion, die einmalig beim Systemstart ausgeführt wird, werden die Verbindungen mithilfe der zuvor erstellten Objekte erstellt. Das Wifi-Objekt wird mit dem Netzwerk des Raspberry Pi verbunden. Dann kann über das MQTT-Objekt eine Verbindung zu dem Broker hergestellt werden. Mithilfe des NFC-Objekts wird überprüft ob die Verbindung zum PN532-Sensor über SPI hergestellt werden konnte.

Das MQTT-Objekt wird außerdem dazu genutzt eine Callback-Funktion zu registrieren, mithilfe dieser der Informationsgehalt der empfangenen Nachrichten in die globale Zustands-Variable geschrieben wird. Zuletzt wird das Objekt genutzt, um ein Topic zu abonnieren. An der Werkerassistenzstation wird das Topic ”rfid/assembling_mcu_receive” abonniert [siehe Listing 4.2].

Listing 4.2: Abonnieren des MQTT-Topics in der Setup-Funktion

```
1  // subscribe to a topic
2  mqttClient.subscribe(topic_receive);
```

Zustandsverarbeitung und Hauptschleife

Die Hauptschleife folgt auf allen Mikrocontrollern der selben Logik wurde aber jeweils an die individuellen Anforderungen angepasst. Beispielsweise wird an der Werkeras-

sistenzstation die UID eines Tags vergeben werden, falls es noch nicht geschehen ist. Ein Codeausschnitt dieser Station - explizit die Loop-Funktion - ist im Anhang festgehalten [siehe Listing B.1].

Die Hauptschleife prüft kontinuierlich, ob ein NFC-Tag im Lesebereich erkannt wurde [siehe Abbildung 4.1]. Wird ein Tag erfasst, so wird die UID gelesen und die relevanten Speicherblöcke geschrieben [siehe Listing 4.3]. Wenn alle Informationen erfolgreich geschrieben wurden, wird eine Nachricht via MQTT mit den neuen Zuständen an den Broker gesendet.

Listing 4.3: Funktion zum Schreiben einer Information auf einen Speicherblock am Beispiel der Werkerassistenzstation

```
1  success = nfc.mifareultralight_WritePage(ASSEMBLING_BLOCK,
     io_state_temp);
```

Für den Fall, dass eine Nachricht vom Broker empfangen wurde und ein Tag im Empfangsbereich ist, wird die erhaltene Information in den entsprechenden Speicherbereich geschrieben. Das erfordert jedoch, dass ein Tag an einer Station etwas länger verweilt. Diese Zeit wird festgehalten und wenn der Tag dann schließlich entfernt wird, wird eine erneute MQTT-Nachricht gesendet, die über die Weiterfahrt zur nächsten Station informiert.

Diese Logik wird fortlaufend in einer Schleife durchlaufen.

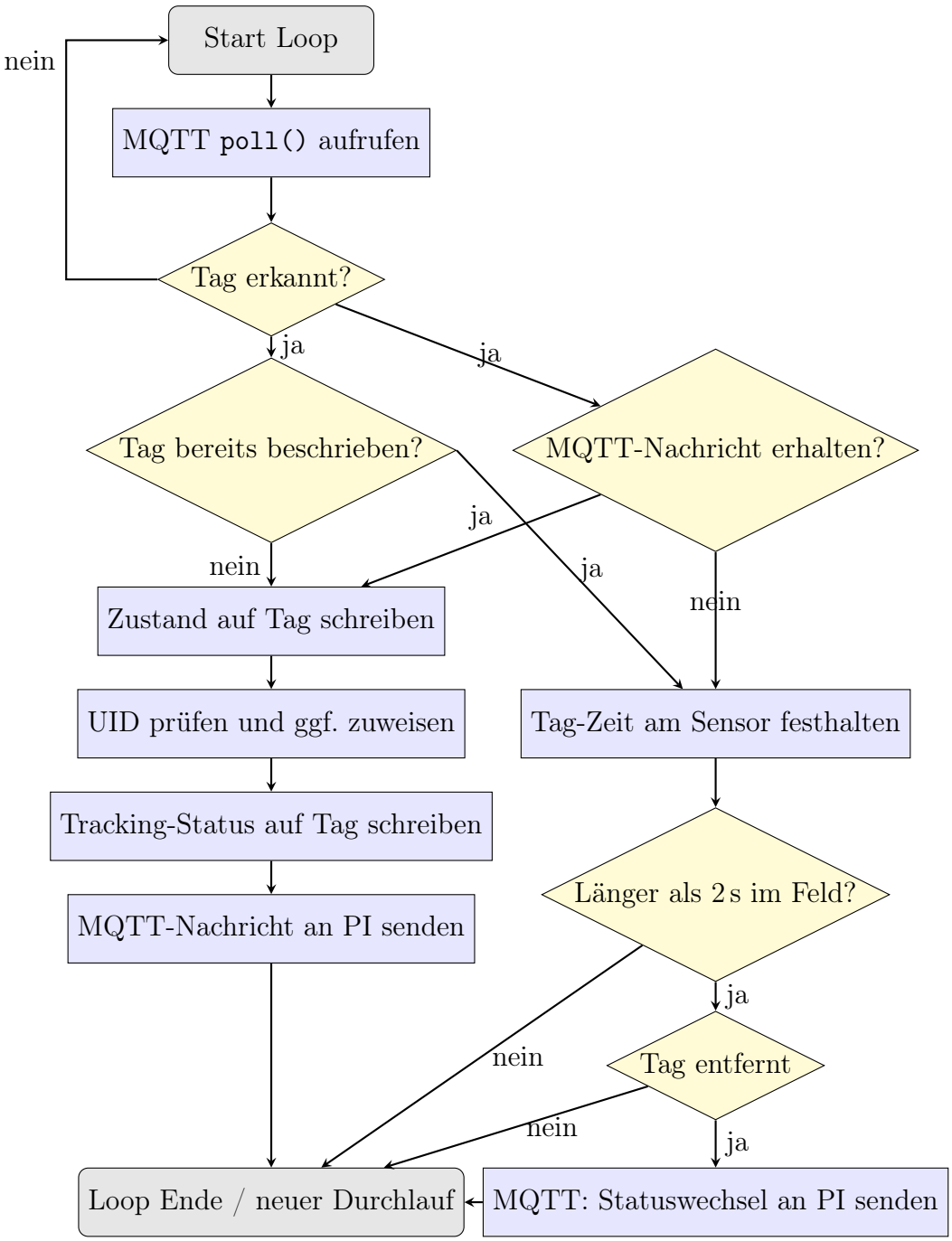

Abbildung 4.1: Ablaufdiagramm der Hauptlogik auf dem Mikrocontroller

Kommunikation über MQTT

Die Kommunikation mit dem Dashboard erfolgt mithilfe eines Raspberry Pi über das MQTT-Protokoll. Jeder Mikrocontroller veröffentlicht Zustandsänderungen auf einem dedizierten Topic wie `rfid/shelf_mcu_send`. Die Daten werden im JSON-Format übertragen [siehe Listing 4.4].

Listing 4.4: Beispiel einer MQTT JSON Nachricht von der Werkerassistenzstation (0x04: zur Schraubstation, 1: iO) (siehe Tabelle 3.6)

```
1  {
2    "uid": "AB CD EF 02",
3    "tracking": "0x04",
4    "state": "1"
5  }
```

Werden Daten für ein abonniertes Topic empfangen, wird in die registrierte Callback Funktion [siehe Listing 4.5] gesprungen und der erhaltene neue Zustand wird in die globale Zustands-Variable geschrieben. Zudem wird eine Variable gesetzt mit der dann in der Loop-Funktion erkannt wird, dass eine Nachricht empfangen wurde[siehe Abbildung 4.1]. Dadurch kann der neue Zustand, durch Zugriff auf die globale Variable in den Speicher des sich im Erkennungsbereich befindlichen NFC-Tags geschrieben werden.

Listing 4.5: Registrierung des Callbacks in der Setup-Funktion

```
1  // set the message receive callback
2  mqttClient.onMessage(onMqttMessage);
```

4.2 Visualisierung der Prozessdaten in Node-RED

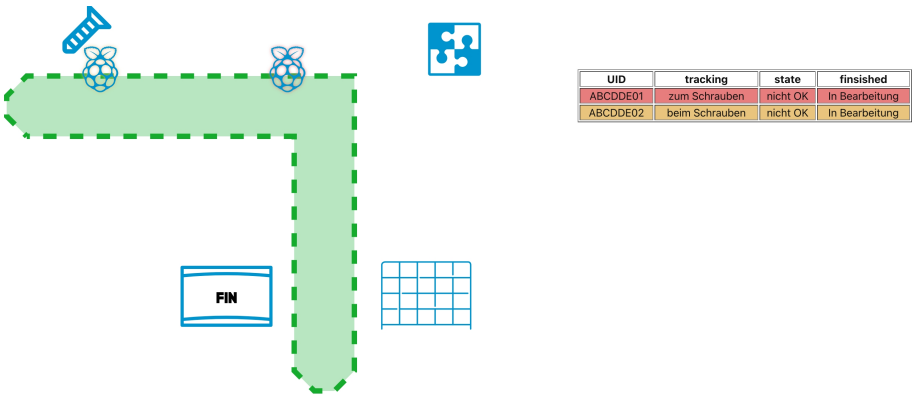

Abbildung 4.2: Node-RED-Dashboard des NFC-Tracking-Systems

Zur Visualisierung der Produktpositionen und -zustände innerhalb des Bandumlaufsystems wurde ein interaktives Dashboard mit Node-RED implementiert (siehe Abbildung 4.2). Dieses Dashboard stellt die vier Stationen geometrisch entsprechend ihrer Anordnung in der Lernfabrik dar. Produkte werden symbolisch als Raspberry-Pi-Icons dargestellt und entlang des virtuellen Förderbands basierend auf ihrem erfassten Tracking-Zustand positioniert.

Die aktuelle Systemübersicht umfasst zwei Hauptkomponenten:

- **Visuelle Darstellung des Förderbands:** Produkte bewegen sich entsprechend ihres Zustands zwischen den Stationen.

- **Tabelle mit Produktinformationen:** Für jedes Produkt wird eine eindeutige UID sowie der aktuelle Tracking-Zustand, der letzte Bearbeitungsstatus und ein Fertigstellungsindikator angezeigt. Die Zeilen sind farblich codiert, um die visuelle Unterscheidbarkeit der Produkte zu erhöhen.

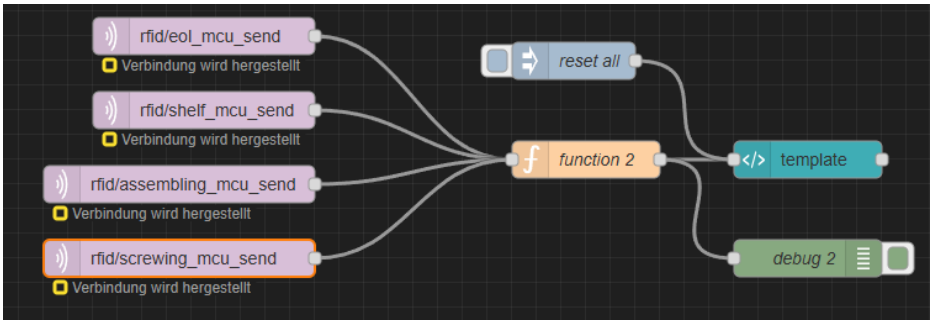

Abbildung 4.3: Node-RED-Flow zum Empfang von Nachrichten

Node-RED ermöglicht eine visuelle und modulare Abbildung der Prozesslogik. Die Empfangslogik ist in Abbildung 4.3 dargestellt. Dabei werden vier verschiedene MQTT-Topics abonniert, über die jeweils eine der vier Stationen Nachrichten an das zentrale System sendet. Die empfangenen JSON-Daten werden über `MQTT-IN`-Nodes verarbeitet und anschließend von `Function`-Nodes analysiert. Die Logik in diesen Funktionsbausteinen umfasst:

- Zuordnung der Nachricht zur entsprechenden Station
- Berechnung der exakten Position auf dem Dashboard
- Generierung einer individuellen Tabellenfarbe pro UID

Zusätzlich enthält der Flow einen Reset-Node zum Zurücksetzen des Dashboards sowie einen Debug-Node zur Laufzeitanalyse.

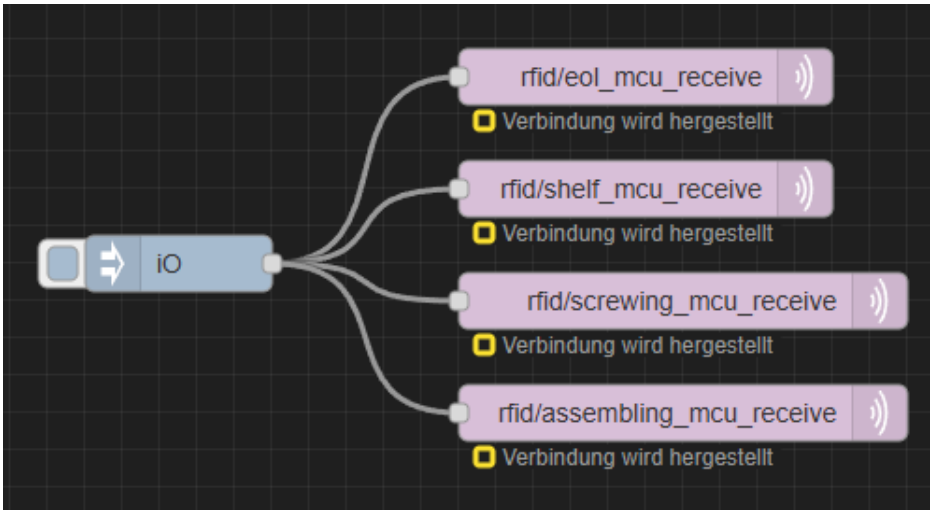

Abbildung 4.4: Node-RED-Flow zum Senden von Nachrichten an Stationen

Für Testzwecke wurde ein dedizierter Sende-Flow implementiert (Abbildung 4.4). Dieser ermöglicht das manuelle Senden von Steuerinformationen an die Mikrocontroller an den Stationen über vier weitere MQTT-Topics. Diese haben dieselbe Namenskonvention wie die Empfangs-Topics, enthalten jedoch den Suffix `_receive` anstelle von `_send`. So kann eine einfache manuelle Steuerung einzelner Produktzustände direkt aus dem Dashboard heraus erfolgen.

Tabelle 4.1: MQTT-Topics

Topic	Beschreibung
$rfid/eol\_mcu\_send$	Datenübertragung von der EOL-Station zum Dashboard
$rfid/shelf\_mcu\_send$	Datenübertragung von der Lagerstation zum Dashboard
$rfid/assembling\_mcu\_send$	Datenübertragung von der Montage-Station zum Dashboard
$rfid/screwing\_mcu\_send$	Datenübertragung von der Schraubstation zum Dashboard
$rfid/eol\_mcu\_receive$	Steuerbefehl an die EOL-Station
$rfid/shelf\_mcu\_receive$	Steuerbefehl an die Lagerstation
$rfid/assembling\_mcu\_receive$	Steuerbefehl an die Montage-Station
$rfid/screwing\_mcu\_receive$	Steuerbefehl an die Schraubstation

5 Zusammenfassung

Auf zwei bis drei Seiten soll auf folgende Punkte eingegangen werden:

- Welches Ziel sollte erreicht werden
- Welches Vorgehen wurde gewählt
- Was wurde erreicht, zentrale Ergebnisse nennen, am besten quantitative Angaben machen
- Konnten die Ergebnisse nach kritischer Bewertung zum Erreichen des Ziels oder zur Problemlösung beitragen
- Ausblick

In der Zusammenfassung sind unbedingt klare Aussagen zum Ergebnis der Arbeit zu nennen. Üblicherweise können Ergebnisse nicht nur qualitativ, sondern auch quantitativ benannt werden, z. B. „. . . konnte eine Effizienzsteigerung von 12 % erreicht werden.“ oder „. . . konnte die Prüfdauer um 2 h verkürzt werden“.

Die Ergebnisse in der Zusammenfassung sollten selbstverständlich einen Bezug zu den in der Einleitung aufgeführten Fragestellungen und Zielen haben.

Literaturverzeichnis

[Fin23] Klaus Finkenzeller. *RFID Handbuch: Grundlagen und praktische Anwendungen kontaktloser Identifikation*. 7. Aufl. Zugriff am 18.05.2025. Hanser Verlag, 2023. ISBN: 978-3-446-43943-6. URL: http://rfid-handbook.de/downloads/G6E_20120413_212413155-24_978-3-446-42992-5_Leseprobe.pdf.

[NFC24] NFC Forum. *Tag Type Technical Specifications*. 2024. URL: https://nfcforum.org/build/specifications#tag-type-technical-specifications (besucht am 19. 05. 2025).

[Nod24] Node-RED Community. *Node-RED – Flow-based programming for the Internet of Things*. Zugriff am 18.05.2025. 2024. URL: https://nodered.org.

[NXP14] NXP Semiconductors. *NTAG213F - NFC Forum Type 2 Tag with Field Detection*. https://www.alldatasheet.com/datasheet-pdf/view/529547/NXP/NTAG213F.html. Datenblatt, Zugriff am 19. Mai 2025. 2014.

[OAS19] OASIS. *MQTT Version 5.0*. https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html. Zugegriffen am 14.05.2025. 2019.

[Sie21] Siemens AG. *SCE Lernpfad: Übersicht der verfügbaren SCE-Lern-/Lehrunterlagen*. Siemens AG, Digital Industries. Nürnberg, Deutschland, Juni 2021. URL: https://www.automation.siemens.com/sce-static/learning-training-documents/classic/guide/00-guide-de.pdf (besucht am 18. 05. 2025).

Verzeichnis verwendeter Formelzeichen und Abkürzungen

EMF	Enhanced Metafile
etc.	et cetera
f.	folgende Seite
ff.	fortfolgende Seiten
JPG	Joint Photographic Experts Group
KI	Künstliche Intelligenz
PDF	Portable Document Format
PNG	Portable Network Graphics
vgl.	vergleiche
z. B.	zum Beispiel

Abbildungsverzeichnis

Tabellenverzeichnis

A Nutzung von Künstliche Intelligenz basierten Werkzeugen

Im Rahmen dieser Arbeit wurden Künstliche Intelligenz (KI) basierte Werkzeuge benutzt. Tabelle A.1 gibt eine Übersicht über die verwendeten Werkzeuge und den jeweiligen Einsatzzweck.

Tabelle A.1: Liste der verwendeten KI basierten Werkzeuge

Werkzeug	**Beschreibung der Nutzung**
ChatGPT	• Grundlagenrecherche zu bekannten Prinzipien bei der Nachrichtenübertragung und Kommunikationsprotokollen (Abschnitt 2.2) • Generierung von Beispielcode bei der Softwareimplementierung auf den Mikrocontrollern (Abschnitt 3.3) • Grammatikalische Überarbeitung von Textpassagen • Rechtschreibprüfung

B Ergänzungen - Ausführung von Quellcode

Listing B.1: Arduino-Loop-Funktion des ESP8266 an der Werkerassistenzstation. Vollständiger Code: GitHub.com

```
1  void loop(void) {
2    // call poll() regularly to allow the library to receive MQTT
     messages and
3    // send MQTT keep alive which avoids being disconnected from the
     broker
4    mqttClient.poll();
5
6    // set write-success variable
7    uint8_t success = false;
8
9    // wait until a tag is found
10   bool currentTagState = nfc.readPassiveTargetID(
     PN532_MIFARE_ISO14443A, uid, &uidLength);
11   if(currentTagState) {
12     if(!start_detached){
13       // handle in-scope timer
14       if (!found_tag){
15         found_tag = true;
16         inscope_start_time = millis();
17       }
18       DEBUG_PRINT("tag found - time in scope [ms]: ");
19       DEBUG_PRINTLN(millis() - inscope_start_time);
20       if (millis() - inscope_start_time >= INSCOPE_TIME){
21         DEBUG_PRINT("INSCOPE_TIME reached after [ms]: ");
22         DEBUG_PRINTLN(millis() - inscope_start_time);
23         DEBUG_PRINTLN("tag is kept at station");
24         found_tag = false;
```

```
25          start_detached = true;
26        }
27      }
28      // handle functionality for a recognized tag
29      if (start_inscope | mqtt_message){
30        start_inscope = false;
31        mqtt_message = false;
32
33        if (uidLength == 7) {
34
35          // processing assembling state
36          success = nfc.mifareultralight_WritePage(ASSEMBLING_BLOCK,
    io_state_temp);
37          if (success) {
38            memcpy(io_state, io_state_temp, sizeof(io_state_temp));
39            print_block("Assembling state", ASSEMBLING_BLOCK, io_state,
    nfc);
40          }
41          else {
42            DEBUG_PRINT("Unable to write to block ");DEBUG_PRINTLN(
    ASSEMBLING_BLOCK);
43          }
44          io_state_temp[3] = 0x00;
45
46          // processing UID
47          success = nfc.mifareultralight_ReadPage(UID_BLOCK, uid_data);
48          if (success) {
49            modify_uid_if_zero(UID_BLOCK, uid_data, nfc);
50          }
51          else {
52            DEBUG_PRINT("Unable to read/write page ");DEBUG_PRINTLN(
    UID_BLOCK);
53          }
54
55          // processing tracking state -> to_shelf
56          success = nfc.mifareultralight_WritePage(TRACKING_BLOCK,
    to_tracking_data);
57          if (success) {
58            print_block("Tracking state", TRACKING_BLOCK,
    to_tracking_data, nfc);
```

```
59          }
60          else {
61            DEBUG_PRINT("Unable to write to page ");DEBUG_PRINTLN(
    TRACKING_BLOCK);
62          }
63
64          // prepare and serialize the data
65          StaticJsonDocument<256> json_data;
66          String jsonString;
67          prepare_data(json_data, uid_data, &io_state[0], &
    at_tracking_data);
68          serializeJson(json_data, jsonString);
69
70          // sending at_assembling via mqtt
71          DEBUG_PRINTLN("send data to PI - in scope");
72          mqttClient.beginMessage(topic_send);
73          mqttClient.print(jsonString);
74          mqttClient.endMessage();
75        }
76        else {
77          DEBUG_PRINTLN("This doesn't seem to be an NTAG2xx tag (UUID
    length != 7 bytes)!");
78        }
79      }
80    }
81    // tag detached
82    else if(start_detached){
83      // prepare and serialize the data
84      StaticJsonDocument<256> json_data;
85      String jsonString;
86      prepare_data(json_data, uid_data, &io_state[0], &to_tracking_data
    [3]);
87      serializeJson(json_data, jsonString);
88      // send to shelf via mqtt
89      DEBUG_PRINTLN("send data to PI - detached");
90      mqttClient.beginMessage(topic_send);
91      mqttClient.print(jsonString);
92      mqttClient.endMessage();
93      // alter global logic
94      start_inscope = true;
```

```
95          start_detached = false;
96        }
97        else {
98          found_tag = false;
99          start_inscope = true;
100         DEBUG_PRINTLN("no tag found");
101       }
102     }
```