SIEMENS

Hochschule München

Fakultät für Elektrotechnik und Informationstechnik

Studiengang Elektrotechnik und Informationstechnik

Industrie 4.0 - Pathfinding auf einer SPS

Bachelorarbeit von Niels Garibaldi

Bearbeitungsbeginn:	xx.xx.2016
Abgabetermin:	xx.xx.2016
lfd. Nr.:	xxxxx

HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN
MÜNCHEN

SIEMENS

Hochschule München

Fakultät für Elektrotechnik und Informationstechnik

Studiengang Elektrotechnik und Informationstechnik

Industrie 4.0 - Pathfinding auf einer SPS

Industry 4.0 - Pathfinding on a PLC

Bachelorarbeit von Niels Garibaldi

betreut von Prof. Dr. K. Ressel

Bearbeitungsbeginn: xx.xx.2016
Abgabetermin: xx.xx.2016
lfd. Nr.: xxxxx

Erklärung des Bearbeiters

1. Ich erkläre hiermit, dass ich die vorliegende Bachelorarbeit selbständig verfasst und noch nicht anderweitig zu Prüfungszwecken vorgelegt habe.

   Sämtliche benutzte Quellen und Hilfsmittel sind angegeben, wörtliche und sinngemäße Zitate sind als solche gekennzeichnet.

2. Ich erkläre mein Einverständnis, dass die von mir erstellte Bachelorarbeit in die Bibliothek der Hochschule München eingestellt wird. Ich wurde darauf hingewiesen, dass die Hochschule in keiner Weise für die missbräuchliche Verwendung von Inhalten durch Dritte infolge der Lektüre der Arbeit haftet. Insbesondere ist mir bewusst, dass ich für die Anmeldung von Patenten, Warenzeichen oder Geschmacksmustern selbst verantwortlich bin und daraus resultierende Ansprüche selbst verfolgen muss.

München, den xx.xx.2016, _____

Niels Garibaldi

Zusammenfassung

Zusammenfassungstext hier einfügen

Abstract

Insert abstract text here

Inhaltsverzeichnis

Abkürzungsverzeichnis

AWL Anweisungsliste

CPPS Cyber-physisches Produktionssystem

CT Corporate Technologies

D* Focussed Dynamic A*

DB Datenbaustein

FB Funktionsbaustein

FC Funktion

FTF Fahrerloses Transportfahrzeug

FTS Fahrerloses Transportsystem

FUP Funktionsplan

IoT Internet of Things

KOP Kontaktplan

MA* Memory Bounded A*

OB Organisationsbaustein

PLC Programmable Logic Controller

RFID Radio Frequency Identification

RTA* Real-Time A*

SCL Structured Control Language

SPS Speicherprogrammierbare Steuerung

STEP7 STeuerungen Einfach Programmieren Version 7

TCP Transport Control Protocol

TIA-Portal Totally Integrated Automation Portal

UDP User Datagramm Protocol

WLAN Wireless Local Area Network

1

Einführung in das Thema Industrie 4.0

Der Begriff „Industrie 4.0“ ist seit seiner Popularisierung durch die Bundeskanzlerin auf der Hannovermesse 2013 vor allem in den Bereichen Produktion und Fertigung in aller Munde. Da er jedoch je nach Branche unterschiedliche Bedeutungen haben kann, soll als Einführung zunächst erläutert werden, was Industrie 4.0 im Kontext der Automatisierungstechnik bedeutet. Der Begriff beschreibt die vierte industrielle Revolution und die einhergehende Verflechtung von informationstechnisch erhobenen Daten in den Produktionsablauf. Ein interessanter Aspekt ist hier der Bereich der prädiktiven Wartung, bei dem Anhand der Auswertung empirischer Daten mögliche Anlagenausfälle frühzeitig erkannt und behoben werden können. Für den weiteren Verlauf dieser Arbeit ist vor allem das sogenannte Cyber-physisches Produktionssystem (CPPS) von Bedeutung. Dieses besteht aus der Verbindung von einzelnen, dezentralen Objekten wie Produktionsanlagen oder Logistikkomponenten, welche mit eingebetteten Systemen ausgestattet und zudem kommunikationsfähig gemacht werden. Durch eingebaute Sensoren und Aktoren kann die Umwelt erfasst und beeinflusst werden. Mittels der Kommunikationskomponenten können Daten aus der Produktion über ein Netzwerk oder das Internet ausgetauscht, beziehungsweise von entsprechenden Diensten ausgewertet, verarbeitet oder gespeichert werden [1]. Sind mehrere Cyber-physische Produktionssysteme an einem Produktionsprozess beteiligt, unabhängig von ihrem Standort, so spricht man auch von einer Smart Factory. Abschließend ist noch zu erwähnen, dass der Begriff „Industrie 4.0“ vor allem im deutschsprachigen Raum verwendet wird. Im internationalen Kontext werden viele der zentralen Punkte von Industrie 4.0 durch das Konzept des Internet of Things (IoT) abgedeckt.

2

Definition der Anforderungen

2.1 Allgemeine Aufgabenstellung

Diese Arbeit beschäftigt sich mit dem Entwurf und der Realisierung eines CPPS im Modellmaßstab. Die resultierende Anlage soll unter anderem dazu dienen, verschiedene Aspekte von Industrie 4.0 vorzuführen und zu veranschaulichen. Kernpunkte, die dargestellt werden sollen, sind vor allem die Dezentralisierung und Skalierbarkeit der Anlage. Den Rahmen für die Bearbeitung dieser Aufgabe bildet die Fachberatung für Automatisierungstechnik der Siemens AG in München. Um die erwähnten Konzepte demonstrieren zu können, soll die Anlage gemäß ihrer realen Vorbilder bestehen aus Bearbeitungsstationen, an welchen der Bearbeitungsprozess simuliert werden kann, und Werkstückträgern, welche Werkstücke durch die Modellanlage zu den Maschinenplätzen transportieren können. Hauptaufgabe des Modells ist es zu zeigen, wie ein Werkstück seine Fertigung selbst organisiert. Zu Beginn des Fertigungsprozesses wird dem Werkstück ein Fertigungsplan übergeben, anhand welchem es sich durch die Anlage bewegt. Im vorliegenden Fall stellen die Werkstückträger gleichzeitig das Werkstück dar, dass die Produktionsanlage durchfährt. Ebenso werden die Bearbeitungsstationen nur symbolisch dargestellt durch entsprechende Markierungen in der Anlagentopologie. Anhand verschiedener Use-cases und Szenarien soll es nach Fertigstellung der Anlage möglich sein, verschiedene Aspekte von Industrie 4.0 anschaulich darzustellen, beispielsweise zu Vorführzwecken bei Kunden.

2.2 Aufteilung der Themenbereiche

Wie beschrieben konzentriert sich die Aufgabe vor allem auf die Entwicklung und Implementierung der Werkstückträgerfahrzeuge. Da dies jedoch ein recht

allgemein gefasstes Themengebiet ist und der zur Realisierung notwendige Aufwand nur schwer abschätzbar ist, wurde entschieden, die Hauptaufgabe in zwei Teilaufgaben zu unterteilen. Diese beiden Themenblöcke sind „Autonomes Fahren in industriellen Transportsystemen“ und „Dynamische Wegfindung in einer flexiblen Produktionsanlage“ [2].Die Dokumentation der Implementierung der dynamischen Wegfindung ist Hauptbestandteil der vorliegenden Arbeit. Den Teil des autonomen Fahrens mittels eines Fahrerloses Transportsystem (FTS) wird im Rahmen einer weiteren Bachelorarbeit behandelt [3] und deshalb nur kurz umrissen.

2.2.1 Fahrerloses Transportsystem

Hauptaufgabe der Werkstückträger, die durch ein Fahrerloses Transportfahrzeug (FTF) realisiert werden sollen, ist es, einen Weg anhand einer vorgegebenen Route ab zu fahren und somit die Werkstücke zwischen den einzelnen Bearbeitungsschritten zur nächsten Station oder Maschine zu transportieren. Herausforderungen hierbei sind die Spurführung und Navigation innerhalb der Anlagentopologie, da die einzelnen FTF nicht wie sonst üblich schienengebunden sind, sondern sich frei durch die Anlage bewegen können. Da sich zudem mehrere Fahrzeuge gleichzeitig in der Anlage befinden können, muss durch geeignete Sensorik eine bevorstehende Kollision erkannt und verhindert werden.

2.2.2 Dynamische Wegfindung

Für die Wegfindung ist es wichtig, dass das Werkstück die Ablaufreihenfolge der benötigten Arbeitsschritte kennt. Diese sollen in einer Art Rezept festgehalten werden und als Grundlage für die Berechnung der Teilrouten zu den Bearbeitungsstationen dienen. Für die Bestimmung der Routen wird zudem eine geeignete Beschreibung der Anlagentopologie benötigt. Innerhalb der Anlage können mehrere Bearbeitungsstationen die gleiche Funktionalität liefern. In diesem Fall soll die Wegfindung eine geeignete Station auswählen können. Diese sollte universell gehalten sein, damit sie auch von dem FTS genutzt werden kann. Die Wegfindung soll so realisiert sein, dass sie auch auf einer der leistungsschwächeren Steuerungen[1] der Siemens S7-1200er Reihe lauffähig ist. Grund hierfür ist unter anderem die erhöhte Ausfallsicherheit, die eine solche Dezentralisierung mit sich bringt. Falls dies nicht realisierbar ist, sollen so wenig Komponenten wie möglich auf ein PC-System ausgelagert werden.
Bei der Berechnung von Routen sollen zudem folgende dynamische Ereignisse berücksichtigt werden:

[1] Speicherprogrammierbare Steuerung (SPS)

- Persistente Blockaden und länger andauernde, nicht permanente Änderungen der Standard-Anlagentopologie.
- Dynamische Blockaden von Teilstrecken durch andere Fahrzeuge zur Kollisionsvermeidung.

2.3 Erwartete Ziele für die Wegfindung

Da bei der Planung des Arbeitsauftrags noch nicht absehbar war, wie zeitaufwendig die Realisierung der Aufgabe sein würde, wurde die Funktionalität der Wegfindungskomponente der Anlage in mehrere Stufen unterteilt:

Grundstufe: Es können detaillierte Teilrouten generiert werden auf der Basis von Anfangs- und Endpunkten.

Stufe 1: Es können komplette Routen bestehend aus Teilrouten anhand eines einzigen Fertigungsplans generiert werden. Zudem existiert pro Funktionalität nur eine Bearbeitungsstation. Mehrere FTF erhalten die gleiche Route und reihen sich hintereinander ein.

Stufe 2: Es werden komplette Routen anhand eines einzigen Fertigungsplans generiert, pro Funktionalität existieren mehrere Bearbeitungsstationen. Die FTF können unterschiedliche Wege nehmen und sich gegenseitig überholen.

Stufe 3: Analog zu Stufe 2, es können jedoch im laufenden Betrieb Störungen in der Form von persistenten Blockaden auftreten, die bei der Wegfindung berücksichtigt werden.

Stufe 4: Analog zu Stufe 3, verschiedene Fahrzeuge können unterschiedliche Fertigungspläne und somit andere Routen verfolgen.

Als zusätzliches Ziel wurde die Visualisierung der geplanten und bereits zurückgelegten Fahrstrecken der verschieden FTF festgehalten, um eine besseren Darstellung der einzelnen Aspekte der definierten Stufen zu ermöglichen. Die Anlage wird zudem in Kooperation mit der Siemens Entwicklungsabteilung Corporate Technologies (CT) entwickelt, welche parallel die gleiche Anlage als Simulation testet und die ermittelten Ergebnisse mit dem Resultat der physikalischen Anlage vergleicht.

3

Theoretische Grundlagen Wegfindung

3.1 Modellierung der Anlagentopologie

Zur Berechnung eines Weges innerhalb der Anlage wird zunächst die Topologie der besagten Anlage benötigt. Für die Funktionsweise der FTF wurde definiert, dass sich alle Fahrzeuge mittels optischer Merkmale auf einer definierten Teilstrecke bewegen. Zudem soll es den Fahrzeugen nur an festgelegten Entscheidungspunkten möglich sein, ihren Fahrzustand zu ändern und eine andere Teilstrecke auszuwählen. Dies bedeutet, dass alle Fahrzeuge, sobald sie sich für eine Teilstrecke entschieden haben, dieser bis zum nächsten Entscheidungspunkt folgen. Auf Basis einer solchen logischen Unterteilung der Anlage in Entscheidungspunkte und Teilstrecken als Verbindungen zwischen zwei solcher Punkte, liegt es nahe als Datenstruktur für die Modellierung der Anlagentopologie einen Graphen zu verwenden. Die Entscheidungspunkte entsprechen hierbei den Knoten und die korrespondierenden Teilwegstrecken stellen die Kanten des Graphen dar. Da sich die FTF möglichst frei durch die Produktion bewegen sollen, wird als Grundform der Anlage ein ungerichteter Graph zur Abbildung der Topologie verwendet, jedoch soll es für die spätere Wegberechnung unerheblich sein, ob es sich um einen gerichteten oder ungerichteten Graphen handelt.

insert graphic about graphs here

Da der Graph die Abbildung einer realen Anlage ist, kann zudem ausgeschlossen werden, dass die Gewichtung der Kanten negativ ist, da dies je nach Art der Gewichtung nur wenig Nutzen bringen würde. Es existieren beispielsweise keine negativen Streckenabstände oder Fahrzeiten, die eine spezielle Betrachtung erforderlich machen und somit die Wahl der Wegfindungsalgorithmen einschränken würden.

3.2 Auswahl der Algorithmen

Es existieren mehrere Algorithmen, welche sich mit der Problemstellung der Berechnung eines kürzesten Pfads zwischen zwei Knoten eines gewichteten Graphen befassen. Um eine passende Vorgehensweise auswählen zu können muss zuerst definiert werden, welche Kriterien ein potentieller Wegfindungsalgorithmus erfüllen muss.

3.2.1 Auswahlkriterien

Für ein geeignetes Wegfindungssystem müssen die folgenden Aspekte berücksichtigt werden:

1. Die Berechnung soll auf einer SPS des niedrigen Leistungssegments durchgeführt werden. Somit müssen die daraus resultierenden Beschränkungen bezüglich Komplexität und Speicherbedarf erfüllt sein.

2. Eine SPS ist ein Echtzeitsystem, welches das Programm zyklisch abarbeitet. Gleichzeitig wird auf der Steuerung auch die Fahraufgabe realisiert. Dadurch muss die Rechenzeit kurz genug sein, um eine Reaktionsfähigkeit des Fahrprogramms sicherstellen zu können.

3. Die Wegfindung soll während der Ausführung eines vorausberechneten Fahrauftrags auf andere Fahrzeuge oder Änderungen der Anlagentopologie reagieren können.

4. Es soll ein optimaler Weg gefunden werden. Optimal bedeutet im vorliegenden Fall einen möglichst kurzen Weg zum Zielknoten.

5. Die Anlage soll gleichmäßig ausgelastet sein in Bezug auf Bearbeitungsstationen mit gleicher Funktionalität.

Eine SPS, wie sie in Punkt 1 beschrieben wird, erlaubt nur eine geringe Schachtelungstiefe der Unterprogrammaufrufe. Die verwendeten Steuerungen der S7-1200er Reihe erlauben beispielsweise eine Schachtelungstiefe von maximal 16 Bausteinaufrufen aus zyklischen und Anlauf-Bausteinen heraus [4]. Da diese kleinen SPS zudem über stark begrenzten Speicherplatz verfügen, muss bei einer Implementierung mittels Stack überprüft werden, dass der hierfür statisch allozierte Speicherbereich auf der Steuerung zur Verfügung steht. Dies beschränkt den Einsatz von rekursiven Algorithmen jedoch auf Anwendungen, bei denen die Anzahl der Rekursionen von vornherein bekannt ist. Ein weiterer Nachteil von Rekusrsion ist, dass das Zeitverhalten von rekursiven Bausteinaufrufen nur schwer abschätzbar ist, und somit vorgegebene harte Echtzeitbedingungen des

Systems verletzt werden können. Da die Anlagen aber beliebiger Art sein sollen, können Algorithmen die Rekursion verwenden somit nicht generell genutzt werden. Zudem fallen für die Implementierung diejenigen Algorithmen heraus, die nur sehr umständlich in der begrenzten Programmierumgebung für Echtzeitsteuerungen realisiert werden können.
Die Einhaltung der in Punkt 2 beschriebenen Echtzeitbedingung ist keine generelle Frage des Algorithmus sonder eine Sache der Implementierung. Hier müssen Programmteile mit Schleifenkonstrukten möglicherweise aufgebrochen und die Berechnung auf verschiedene Zyklen aufgeteilt werden.
Die in Punkt 3 erwähnte Reaktionsfähigkeit erfordert von den Algorithmen, dass sie auch in nur teilweise bekannten Umgebungen sicher einen Weg finden können. Für das Zusammenspiel ist eine Methode für die Vorhersage der Positionen anderer Fahrzeuge notwendig.
Da aus Gründen der Anlageneffizienz die Laufzeit eines Werkstücks durch die Anlage möglichst minimiert werden soll, wird von dem Wegfindungsalgorithmus gemäß Punkt 4 erwartet, dass er nicht nur schnell einen möglichen Weg findet, sondern dass der gefundene Weg auch der kürzeste, mit den begrenzten Anlageninformationen[1] berechenbare, Pfad zum Zielknoten ist.
Da die im Punkt 5 erwähnte gleichmäßige Auslastung mehr mit der Auswahl des geeigneten Zielknotens als mit der eigentlichen Wegfindung zu tun hat, ist diese Anforderung nur begrenzt für die Auswahl der Algorithmen von Bedeutung.

3.2.2 Betrachtete Algorithmen

Im Angesicht der in 3.2.1 ermittelten Auswahlkriterien wurden die folgenden Algorithmen näher betrachtet:

Dijkstra : Einer der Grundalgorithmen für das Kürzeste-Pfad-Problem, bei dem auch die kürzesten Entfernungen aller Knoten zu einem Startknoten ermittelt werden können.

A* : Eine Abwandlung des Dijkstra-Algorithmus, bei dem die Anzahl der betrachteten Knoten verringert werden kann, indem Zusatzinformationen in Form einer Heuristik als Entscheidungshilfe für die Betrachtungsreihenfolge verwendet werden.

[1] Abschnitt 5.2.2 erläutert, dass Bewegungen anderer Fahrzeuge nur begrenzt vorausberechnet werden können.

Memory Bounded A* (MA*)[5]: Dieser Algorithmus beschränkt den Speicherverbrauch indem er den Topologiegraphen in Teilbäume unterteilt und nur erfolgversprechende Knoten und Teilgraphen im Speicher behält.

Real-Time A* (RTA*)[6]: Dieser Algorithmus unterteilt den Hauptalgorithmus in Planungs- und Ausführungsphasen und beschränkt die Anzahl der betrachteten Knoten anhand einer Alpha-Beta-Suche. Durch diese Unterteilung kann ein gewisses Maß an Dynamik gewonnen werden, da der Algorithmus nach jeder Ausführungsphase die Restroute neu evaluiert und gegebenenfalls die aktuelle Route gegen einen neuen optimierten Weg ersetzt.

Focussed Dynamic A* (D*)[7][8]: Dieser Algorithmus wurde entwickelt für Topologien, welche nur teilweise bekannt sind, beziehungsweise die sich dynamisch verändern können. Zeiger die entlang des jeweils kürzesten Weges vom aktuellen Knoten zum Startknoten zeigen werden bei Anlagenänderung durch sogenannte „Modify-Cost-Operationen“ ausgehend von der Anlagenänderung kaskadierend modifiziert bis sich ein neuer statischer Zustand eingestellt hat. Die Wegberechnung nutzt diese Zeiger zur schnelleren Berechnung des kürzesten Pfades.

Der Dijkstra-Algorithmus ist aufgrund seiner langen Berechnungszeit ungeeignet, für die Wegfindung im laufenden Betrieb der FTF jedoch kann ausgenutzt werden, dass bei der Berechnung des kürzesten Pfades via Dijkstra die Abstände aller besuchten Knoten zum Startknoten ermittelt werden. Diese Abstände bilden die Basis zur Verbesserung der bei einem der anderen Algorithmen genutzten Heuristiken. Die Algorithmen MA*, RTA* und D* sind alle Weiterentwicklungen des A*-Algorithmus. Die zu entwickelnde Modellanlage besitzt im vorliegenden Fall nur eine begrenzte Anzahl von Knoten. Dadurch ist der Nachteil des hohen Speicherbedarfs des Grundalgorithmus A* hier vernachlässigbar und der zusätzliche Aufwand für die Implementierung von MA* zunächst ignoriert werden. Bei realen Anlagen mit einer höheren Anzahl an Knoten muss jedoch im Einzelfall betrachtet werden, ob eine Steuerung mit größerem Arbeitsspeicher oder die Implementierung des MA*-Algorithmus notwendig ist. Der D*-Algorithmus wurde wegen seiner höheren Komplexität und der Beschränkungen durch die

Programmierumgebung nicht implementiert. Als zyklischer Wegfindungsalgorithmus wurde eine Mischform des A*-Algorithmus mit der Planungs- und Ausführungsaspekten ähnlich denen von RTA* verwendet, indem in regelmäßigen Abständen der bisher gefundene Teilweg verifiziert und bei Bedarf neu berechnet wird. Die verwendeten Algorithmen sollen nun etwas näher betrachtet werden.

3.3 Gemeinsamkeiten der Algorithmen

Aufgrund der Tatsache, dass es sich bei dem A*-Algorithmus um eine Weiterentwicklung des Dijkstra-Algorithmus handelt, gibt es einige Grundprinzipien, die bei von beiden Algorithmen verwendet werden. Zum einen arbeiten beide mit einer Unterteilung der Graphenknoten in Unterklassen mit definierten Eigenschaften. Zum anderen wählen die Algorithmen den Knoten, der als nächstes betrachtet wird, nach bestimmten Kriterien aus. Bevor die Algorithmen einzeln vorgestellt werden, sollen zunächst ihre Gemeinsamkeiten dargestellt werden.

3.3.1 Klassifizierung der Knoten

Um einen kürzesten Pfad zwischen zwei Knoten eines Graphen zu ermitteln, ist es hilfreich,während der Pfadberechnung alle Knoten des Graphen in drei Klassen aufzuteilen [9]:

Klasse A: Die Menge aller Knoten, zu denen bereits ein kürzester Pfad vom Startknoten aus bekannt ist. Diese Klasse wird auch als Closed-List bezeichnet.

Klasse B: Die Menge aller Knoten, die mit mindestens einem Knoten aus Gruppe A verbunden sind, jedoch selbst nicht zu A gehören. Diese Klasse wird auch als Open-List bezeichnet.

Klasse C: Die Menge der restlichen Knoten, die nicht in den Gruppen A oder B enthalten sind, und somit noch nicht betrachtet wurden.

Analog können auch die Kanten des Graphen klassifiziert werden.

Klasse I: Die Menge aller Kanten, die in einem kürzesten Pfad zu einem der Knoten aus Gruppe A vorkommen.

Klasse II: Die Menge aller Kanten, aus denen die nächste Kante für Gruppe I ausgewählt wird, wenn der korrespondierende Knoten aus B zur Gruppe A hinzugefügt wird. Es existiert immer genau eine derartige Kante für jeden Knoten aus Gruppe B.

Klasse III: Die restlichen Kanten, die sich nicht in den Klassen I und II befinden.

Eine derartige Einteilung der Knoten vereinfacht während der Berechnung des kürzesten Pfades die Entscheidung, welche Knoten momentan für die Wegfindung interessant sind und somit näher betrachtet werden sollten.

3.3.2 Auswahl des nächsten Knotens

Da nun die interessanten Knoten des Graphen identifiziert wurden, stellt sich die Frage in welcher Reihenfolge diese Knoten am besten betrachtet werden sollten. Diese Reihenfolge kann mittels einer Entscheidungsfunktion $f(n)$ bestimmt werden.

- *Seien s, n und z beliebige Knoten des Graphen G, wobei s der Startknoten und z der Zielknoten für die Ermittlung des kürzesten Weges von s zu z sind.*

- *Sei $f(n)$ der Wert eines kürzesten Pfades vom Knoten s zum Knoten z, der zusätzlich den Knoten n enthält.*

Es sei zunächst nicht bekannt, ob der Knoten n auch auf dem kürzesten Pfad von s nach z liegt. Diese Entscheidungsfunktion, kann unterteilt werden in zwei Unterfunktionen $g(n)$ und $h(n)$ für die gilt [11]:

$$f(n) = g(n) + h(n) \tag{3.1}$$

Die Funktion $g(n)$ liefert hierbei den Wert des kürzesten Pfads vom Knoten s zu n. Die Funktion $h(n)$ beschreibt den Wert eines kürzesten Pfades von n zum Ziel z. A priori[2] sind die exakten Werte für $g(n)$ und $h(n)$ aber möglicherweise noch nicht bekannt. Somit werden für die Berechnung zunächst Schätzwerte für $g(n)$ und $h(n)$ verwendet. Sei also $\hat{g}(n)$ der Wert des bisher gefundenen kürzesten Pfades von s nach n und sei $\hat{h}(n)$ der mittels einer Heuristik geschätzte Wert des

[2] also bevor der Wert von $f(n)$ bestimmt wurde.

kürzesten Pfads von n nach z. Somit kann der Wert von $f(n)$ geschätzt werden durch:

$$\hat{f}(n) = \hat{g}(n) + \hat{h}(n) \tag{3.2}$$

Man geht zunächst davon aus, dass sich n in der Klasse B befindet. s befindet sich per Definition des Startknotens bereits in A, da die Entfernung von s zu s gleich null ist. Gemäß der Definition der Klasse B existiert somit mindestens ein Knoten k_i aus der Klasse A, der durch eine Kante mit dem Knoten n verbunden ist und für den der $g(k_i)$ bereits bestimmt wurde[3]. Sei $v_{k_i,n}$ die Gewichtung der Verbindungskante des Knotens k_i nach n, so kann der exakte Wert für $g(n)$ wie folgt bestimmt werden:

$$\hat{g}(n) = g(k_i) + min(v_{k_i,n}) = g(n) \tag{3.3}$$

Dadurch kann der exakte Wert für $g(n)$ bestimmt werden. Die Kante mit der kleinsten Gewichtung, die den Knoten n mit einem Knoten k_i aus A verbindet, gehört zu der in Abschnitt 3.3.1 definierten Klasse II.
Es kann beweisen werden [11], das die Wegfindungsalgorithmen immer den kürzesten Pfad von s nach z finden, sollte dieser existieren, wenn die Heuristik „zulässig“ ist. Eine zulässige Heuristik, ist eine Schätzung deren Wert kleiner ist als der tatsächliche Wert, der geschätzt werden soll. Im vorliegenden Fall bedeutet dies, dass gelten muss:

$$\hat{h}(n) \leq h(n). \tag{3.4}$$

Dies hat zur Folge, dass der kürzeste Pfad von s nach z unter Verwendung einer solchen Heuristik $\hat{h}(n)$ bei Beendigung des Algorithmus nur aus Knoten der Klasse A besteht, also keine anderer Pfad existiert, der kürzer ist als der gefundene. Durch Speicherung des jeweiligen Vorgängers des aktuellen Knotens kann nach Beendigung der Algorithmen der kürzeste Pfad konstruiert werden.
Der Dijkstra-Algorithmus stellt hier den Standardfall dar, bei dem $\hat{h}(n) = 0$ angenommen wird. Im Bezug auf A* werden unterschiedliche Werte für die Heuristik in Abschnitt 3.5.2 näher betrachtet.

3.4 Dijkstra-Algorithmus

Der erste für die Wegfindung verwendeten Algorithmus ist wie bereits erwähnt, der nach seinem Erfinder benannte Dijkstra-Algorithmus, der den kürzesten Pfad

[3] dies ist durch die Definition von A gegeben.

zwischen einem Startknoten und einem Endknoten[4] eines Graphen ermittelt. Er kann verwendet werden für beliebige positiv gewichtete, gerichtete oder ungerichtete Graphen. Da in Abschnitt 3.1 negative Kantengewichtungen ausgeschlossen wurden, lässt sich der Algorithmus ohne Abwandlung auf den Graphen der Anlagentopologie anwenden.

3.4.1 Grundprinzip

Der Algorithmus funktioniert wie folgt. Zu Beginn befinden sich alle Knoten und Kanten respektive in den Klassen C oder III, da sie noch nicht betrachtet wurden. Als erstes wird der Startknoten S zur Klasse A hinzugefügt, da ein kürzester Pfad vom Startknoten zu sich selbst mit der Entfernung null bekannt ist. Der Startknoten kann somit als geschlossen angesehen werden. Nun werden folgende Schritte solange wiederholt, bis der Zielknoten erreicht wurde:

1. Man betrachte alle Kanten z die den soeben zu A hinzugefügten Knoten mit einem Knoten K aus B oder C verbinden.
   - Gehört K bereits zur Gruppe B, so wird untersucht ob die Kante z zu einem kürzeren Pfad zu K führt, als der bisher gefundene kürzeste Pfad. Ist dies der Fall, so ersetzt z die korrespondierende Kante aus II. Die Kante die durch z ersetzt wird, wird wieder zur Gruppe III hinzugefügt.
   - Gehört K zur Gruppe C, so wird K zu B und z zu II hinzugefügt.
2. Es wird die in Abschnitt 3.3.2 vorgestellte Funktion $f(n)$ verwendet, um genau den Knoten B_i in der Klasse B zu finden, der den kleinsten Wert für $f(B_i)$ besitzt. Da bei dem Dijkstra-Algorithmus $\hat{h}(n)$ gleich null ist, wird genau der Knoten B_i ausgewählt, für den $g(B_i)$ den kleinsten Wert hat, der also von den offenen Knoten den kleinsten Abstand zum Startknoten hat. Die zu B_i gehörende Kante aus II wechselt damit in die Klasse I.

3.4.2 Darstellung der Funktionsweise

Die **insert graphic here** zeigt die Funktionsweise des Algorithmus an einem einfachen Beispielgraphen mit vier Knoten.

[4] oder allen anderen Knoten.

3.4.3 Berechnungsaufwand

Der Berechnungsaufwand des Dijkstra-Algorithmus beträgt in Landau-Notation vereinfacht $\mathcal{O}(|\mathcal{V}|^2)$, beziehungsweise $\mathcal{O}(|\mathcal{V}|\log|\mathcal{V}| + |\mathcal{E}|)$ bei der Implementierung mittels Fibonacci-Heaps [10], wobei $|\mathcal{V}|$ gleich der Anzahl der Knoten und $|\mathcal{E}|$ gleich der Anzahl der Kanten des Graphen ist. Diese Vereinfachung gilt jedoch nur, wenn die Anzahl der Kanten in der gleichen Größenordnung liegt wie die Anzahl der Knoten, also beispielsweise nicht für vollständige graphen, für die gilt $|\mathcal{E}| = \mathcal{O}(|\mathcal{V}|^2)$.

3.5 A*-Algorithmus

Der zweite verwendete Algorithmus für die Wegfindung ist der sogenannte A*-Algorithmus. Dieser ist eine Weiterentwicklung des in 3.4.1 vorgestellten Dijkstra-Algorithmus. Der grundlegende Unterschied ist jedoch, dass bei A* eine andere Bewertung der Betrachtungsreihenfolge von Knoten bei der Berechnung verwendet werden. Dies bedeutet, dass diejenigen Knoten und Kanten zuerst expandiert werden sollen, bei denen die Wahrscheinlichkeit höher ist, das sie Teil des kürzesten Pfades zum Zielknoten sind. Im Vergleich dazu wurde beim Dijkstra-Algorithmus immer genau der Knoten aus B als nächstes expandiert, der den kürzesten Abstand zu dem Startknoten hatte**Insert graphic dijkstra vs A***. Um eine Abschätzung der Wahrscheinlichkeit eines Knotens bezüglich seiner Zugehörigkeit zu dem kürzesten Pfad vom Start zum Zielknoten treffen zu können, werden somit zusätzliche Informationen über die Beziehung zwischen einzelnen Knoten und dem Zielknoten benötigt. Diese Bewertung von Knoten ist die im Abschnitt 3.3.2 vorgestellte Heuristik bezeichnet. Mittels dieser Bewertung kann nun bereits vor Erreichen des Zielknotens eine Aussage über die Länge der Route gemacht werden.

3.5.1 Grundprinzip

Die Funktionsweise des A*-Algorithmus kann somit als Erweiterung des Dijkstra-Algorithmus beschrieben werden [11]. Unter Verwendung der in Abschnitt 3.3.2 definierten Entscheidungsfunktion kann wie folgt verfahren werden:

- s, z und n sind Knoten eines Graphen. Es soll der kürzeste Pfad von s nach z ermittelt werden.

1. Man markiere s als „offen“ und berechne $f(s)$.
2. Man wähle für n denjenigen Knoten aus der Open-List aus, der den kleinsten Wert für $f(n)$ besitzt.
3. Falls $n = z$, wird n als „geschlossen“ markiert und der Algorithmus beendet.
4. Andernfalls markiere man n als „geschlossen“ und füge alle von n über maximal eine Kante erreichbaren Knoten n' zur Liste der offenen Knoten hinzu, die noch nicht als geschlossen markiert wurden und berechne $f(n')$. Man springe zurück zu Schritt 2.

Anhand dieser Vorgehensweise ist zu erkennen, dass bei dem A*-Algorithmus diejenigen Knoten bevorzugt betrachtet werden, die durch die Entscheidungsfunktion als „wahrscheinlicher zum Ziel führend“ bewertet wurden, da anders als bei Dijkstra $\hat{h}(n)$ bei der Berechnung von $f(n)$ einen Einfluss auf das Ergebnis hat. Somit ist es sinnvoll diesen Bewertungsvorgang näher zu betrachten.

3.5.2 Vergleich verschiedener Heuristiken h(n)

Die in Kapitel 3.3.2 besprochene Entscheidungsfunktion $f(n)$ und diesbezüglich vor allem auch der Wert von $\hat{h}(n)$ haben einen großen Einfluss auf das Verhalten des A*-Algorithmus. Zur Veranschaulichung sollen einige Möglichkeiten für diese Heuristik betrachtet werden [12]:

1. $\hat{h}(n) = 0$**:** Wenn der Wert der Heuristik konstant auf Null gesetzt wird, so ist dies gleich zu setzen damit, als würden keine Zusatzinformationen zur Berechnung der Route verwendet. Dies bedeutet das immer genau der Knoten aus der Menge von offenen Punkten ausgewählt wird, dessen Abstand zum Startknoten am kleinsten ist. Der kürzeste gefundene Pfad durch einen Zwischenknoten n hat zu jedem Zeitpunkt i immer $f_i(n) = g_i(n)$. Somit entspricht der A*-Algorithmus mit der Heuristik $\hat{h}(n) = 0$ genau dem Dijkstra-Algorithmus aus 3.4.1.
2. $\hat{h}(n) < h(n)$**:** Wenn der Abstand eines Knotens n zum Ziel immer unterschätzt wird, also immer kleiner ist als der tatsächliche Abstand, so findet der A*-Algorithmus, wie bereits im Abschnitt 3.3.2 gezeigt, garantiert den kürzesten Pfad zwischen Start und Ziel. Je kleiner $\hat{h}(n)$ im Verhältnis zu $h(n)$ ist, desto mehr Knoten muss A* expandieren und umso langsamer läuft der Algorithmus.
3. $\hat{h}(n) = h(n)$**:** Wenn der geschätzte Abstand eines Knotens genau dem tatsächlichen Abstand entspricht, expandiert A* immer die minimale Anzahl von

Knoten entlang des kürzesten Pfades. Das heißt, dass wenn der kürzeste Pfad sechs Knoten enthält, der A*-Algorithmus nur genau sechs Knoten expandiert bevor der Zielknoten geschlossen und somit der kürzeste Pfad gefunden wurde. Zudem ist der Wert von $f_k(n)$ für jeden Teilknoten k auf dem kürzesten Pfad immer gleich dem exakten Wert des Abstands zwischen Start und Zielknoten.

4. $\hat{h}(n) > h(n)$**:** Wenn der geschätzte Abstand eines Knotens größer ist als der tatsächliche Abstand zum Zielknoten, so kann nicht mit Bestimmtheit gesagt werden, dass kein kürzerer Pfad zu bereits geschlossenen Knoten existieren. Somit müssten, um den möglicherweise existierenden tatsächlichen kürzesten Pfad zu finden, bereits geschlossene Knoten erneut geöffnet werden. Wird dies nicht getan, so kann der A*-Algorithmus zwar unter Umständen schneller einen Pfad vom Start zum Ziel finden, es ist aber nicht garantiert, dass es sich um den Kürzesten handelt.

5. $\hat{h}(n) \gg h(n)$ Wenn der Schätzwert der Heuristik viel größer ist als der tatsächliche Abstand zum Zielknoten, so ist die Entscheidungsfunktion $f(n)$ nur noch abhängig von $\hat{h}(n)$ und die anderen $g(n)$ kann vernachlässigt werden. Somit hängt die Reihenfolge der expandierten Knoten nur noch von den Werten der Heuristik der entsprechenden Knoten ab.

Anhand der vorgestellten Eigenschaften der Heuristik wird erkennbar, das der A*-Algorithmus durch die Modifizierung der Heuristikberechnung an die gegebenen Anforderungen anpassbar ist. Anstatt einer genauen, aber langsamen Pfadberechnung, kann auf Kosten der Genauigkeit auch schnell ein möglicherweise nicht optimaler Weg zum Ziel gefunden werden. Zudem kann durch perfektes Wissen der Topologie der Anlage in kürzester Zeit der kürzeste Weg ermittelt werden.

3.5.3 Darstellung der Funktionsweise

Anhand eines einfachen Beispiels soll jetzt die Funktionsweise des A*-Algorithmus dargestellt werden. Die Knoten des folgenden Graphen sollen Städte darstellen und die Kanten geben die Verbindungen durch Schnellstraßen oder Autobahnen wieder. Die Gewichtungen der Kanten stellen in diesem Beispiel die Entfernungen zwischen den Städten dar. Als Heuristik wurde hier die Entfernung in Luftlinie zwischen den jeweiligen Städten und der Zielstadt gewählt, da diese die Luftlinie immer kürzer[5] ist als eine Straße zwischen den Städten und somit die Bedingung $\hat{h}(n) \leq h(n)$ erfüllt ist:
insert graphic here

[5] oder im Extremfall gleich lang.

3.5.4 Berechnungsaufwand

Der Berechnungsaufwand für den A*-Algorithmus lässt sich nur schwer festlegen, da wie in 3.5.2 beschrieben, die Berechnung des kürzesten Pfads sehr stark von der Heuristik abhängt. Unter der Annahme, dass auch wirklich der kürzeste Pfad gefunden wird, muss für die Heuristik gelten:

$$0 \leq \hat{h}(n) \leq h(n) \tag{3.5}$$

Somit muss auch der Berechnungsaufwand zwischen dem des Dijkstra-Algorithmus und dem Aufwand im Falle einer exakten Heuristik liegen. Da bei einer exakten Heuristik die Zahl der expandierten Knoten nur von der Anzahl der Knoten auf dem kürzesten Pfad abhängt, ist die Berechnungsaufwand konstant. Daraus folgt für eine beliebige Heuristik, die Bedingung 3.5 erfüllt:

$$\mathcal{O}(|\mathcal{V}|^2) \geq \mathcal{O}(\hat{h}(n)) \geq \mathcal{O}(|\mathcal{V}|) \tag{3.6}$$

Somit ist der Berechnungsaufwand im schlechtesten Fall quadratisch, im besten Fall aber gleich der Anzahl der Knoten des kürzesten Pfads. Eine bessere Heuristik führt somit dazu, dass weniger Knoten als im schlechtesten Fall expandiert werden müssen und sich somit der Rechenaufwand verringert.

3.6 Zusammenspiel der Algorithmen

Wie bereits in Abschnitt 3.2.2 erwähnt, wird nun ausgenutzt, dass der Dijkstra-Algorithmus die Abstände vom Startknoten zu allen betrachteten Knoten ermittelt. Der Nachteil, dass er vergleichsweise langsam arbeitet wird umgangen, indem der Algorithmus nicht in der zeitkritischen zyklischen Bearbeitungsphase sondern beim Hochfahren der CPU ausgeführt wird. Hier ist die Fahrzeugsteuerung[6] noch nicht aktiv und somit können auch längere Berechnungen ausgeführt werden. Zusätzlich werden nicht alle Knoten als Startknoten für die Abstandsberechnung mittels Dijkstra verwendet, sondern nur solche, die im späteren Anlagenbetrieb als endgültige Zielknoten für die Wegfindung infrage kommen.Als solche Zielknoten kommen allem die Knoten in Frage, an denen sich Maschinen befinden oder die eine spezielle Funktion, wie beispielsweise der Anlagenausgang, erfüllen. Transitknoten, die nur im Verlaufe der Abarbeitung einer Route durchfahren werden, aber an denen nie eine Route endet, können hierbei ausgelassen werden.
Die Ergebnisse der Dijkstra-Berechnungen werden in einer Tabelle gesichert und als Basis für die Heuristik der Wegfindung mittels A* im zyklischen Betrieb ver-

[6] und vor allem auch der Watchdog-Timer

wendet. Wie in Abschnitt 3.5.2 gezeigt wurde, ist die Anzahl der expandierten Knoten, und somit auch die Laufzeit, minimal, wenn als Heuristik der exakte Abstand zum Ziel genutzt wird. Dieser Fall ist, unter der Bedingung das sich die Anlage seit dem Hochfahren der CPU nur geringfügig geändert hat, durch die Verwendung der Abstände des entsprechenden Ziels aus der Dijkstra-Tabelle gegeben. Auch bei Änderungen, wie beispielsweise Sperrungen von Kanten oder der Blockade von Teilstrecken durch andere Fahrzeuge, erfüllt diese Heuristik noch die Anforderung $\hat{h}(n) \leq h(n)$, da die neue exakte Route sicher länger ist, als die vorausberechnete ideale Route ohne Hindernisse. Bei Hinzufügen neuer Kanten zur bestehenden Anlage muss die CPU neu gestartet werden[7], da hierdurch kürzere Wege entstehen könnten, welche die Bedingung aus Gleichung 3.5 verletzen würden.

[7] aus im Abschnitt 4.2.3 beschriebenen Gründen

4

Technische Implementierung der Algorithmen

4.1 Kurze Einführung in die Programmiersprache und Programmierumgebung

Für die Implementierung der Wegfindung wurde im Kapitel 2.2.2 definiert, dass diese auf einer Siemens SPS (englisch: Programmable Logic Controller (PLC)) der S7-1200er Reihe lauffähig sein muss. Bei der 1200er Reihe handelt es sich um Steuerungen des niedrigen Leistungssegments. Für die Projektierung und Programmierung von Anlagen mit Steuerungen dieser Art wird deine proprietäre Entwicklungsumgebung namens Totally Integrated Automation Portal (TIA-Portal) von Siemens zur Verfügung gestellt.

4.1.1 Siemens TIA-Portal

Das Siemens TIA-Portal vereinigt viele Aspekte der Projektierung von Anlagen in einer einheitlichen Oberfläche. Innerhalb des TIA-Portals können beispielsweise Projekte bestehend aus mehreren Antrieben und SPSen gemeinsam geplant und erstellt werden. Das TIA-Portal ist aktuell in der Version 13 verfügbar und bietet vor allem eine anwenderfreundliche Oberfläche für komplexe Automatisierungsaufgaben. Die Kernkomponente für die Erstellung von Programmen für Speicherprogrammierbare Steuerungen ist die Programmierumgebung STEP7[1]. Abbildung **INSERT GRAPHIC HERE** zeigt die Projektansicht des TIA-Portals

[1] STeuerungen Einfach Programmieren Version 7 (STEP7)

4.1.2 Programmierumgebung STEP7

Die Grundelemente eines STEP7-Projekts sind die projektierten Steuerungen. Diese sind wiederum unterteilt in Teilelemente wie die Hardware-Konfiguration der Steuerung, das Anwenderprogramm, die verwendeten Variablen, benötigte Datentyp-Definitionen und Komponenten zur Überwachung und Modifizierung der Steuerungsdaten im laufenden Betrieb **Insert grafic here**. Für die Implementierung der Wegfindungsalgorithmen sind vor allem das Steuerungsprogramm und die darin verwendeten Datentypen von Bedeutung. Ein Anwenderprogramm besteht aus bis zu vier Arten von Programmbausteinen[13]:

Organisationsbaustein (OB): Diese Bausteine bilden die Schnittstelle zwischen dem Betriebssystem der Steuerung und dem Anwenderprogramm. Sie haben jeweils vordefinierte Funktionalitäten und bilden somit das Grundgerüst des Anwenderprogramms. Die in dieser Implementierung verwendeten OBs sind zum einen der Systemstart-Baustein sowie Bausteine zur zyklischen Abarbeitung von Teilschichten des Programms.

Datenbaustein (DB): Datenbausteine dienen zur Speicherung von variablen Daten, auf die von dem gesamten Anwenderprogramm aus zugegriffen werden kann. Sie werden unter anderem zur Sicherung der Topologiedaten der Anlage, sowie als Schnittstellen zwischen verschiedenen Programmschichten verwendet.

Funktion (FC): Funktionen sind Bausteine zur elementaren Kapselung von Funktionalitäten. Sie werden im Anwenderprogramm definiert als Unterprogramme, die keinen eigenen Speicher zur Sicherung von Variablenwerten zwischen zwei aufeinanderfolgenden Programmaufrufen benötigen.

Funktionsbaustein (FB): Funktionsbausteine realisieren wie FCs Unterprogramme, stellen aber zusätzlichen Speicherbereich für die permanente Sicherung von internen Variablen zur Verfügung. Bei der Verwendung eines FBs wird bei dessen Initialisierung ein entsprechender Instanz-DB generiert, in welchem Daten für die Verwendung in späteren Programmaufrufen gespeichert werden können.

FCs und FBs entsprechen den Funktionsdefinitionen in anderen Programmiersprachen. Es können die Schnittstellen der Bausteine sowie deren Schnittstellentypen definiert werden. IN-Variablen werden beispielsweise nur lesend verwendet, OUT-Variablen werden nur schreibend und INOUT-Variablen werden sowohl schreibend als auch lesend verwendet. Innerhalb eines Bausteins können zum einen temporäre als auch statische[2] Variablen zur Zwischenspeicherung von Variablenwerten während der Programmabarbeitung genutzt werden. Da statische Variablen einen Instanz-DB benötigen, sind sie nur in FBs verwendbar.
Die Bausteine können in einer von vier Programmiersprachen geschrieben werden. Funktionsplan (FUP) und Kontaktplan (KOP) sind Sprachen zur graphischen Programmierung. Anweisungsliste (AWL) ist eine Assembler-ähnliche Sprache für generelle Programmieraufgaben, welche unter anderem die byteweise Manipulation von Daten vereinfacht. Structured Control Language (SCL) ist eine Pascal-ähnliche Programmiersprache, die durch ihre einfachen Implementierungsmöglichkeiten von Schleifen geeignet ist für die Programmierung komplexer Aufgabenstellungen **insert comparison graphic here**. Bei der Erstellung des Anwenderprogramms für die Wegfindung wurden die verwendeten OBs in FUP erstellt und alle anderen Bausteine in SCL.

4.1.3 Arbeitsweise einer SPS

Eine Speicherprogrammierbare Steuerung arbeitet nach dem Prinzip eines Echtzeitsystems. Das projektierte Anwenderprogramm wird in einer Endlosschleife zyklisch abgearbeitet. Zu Beginn eines Bearbeitungszyklus wird ein Prozessabbild aller Eingangsbaugruppen der Steuerung generiert, welches für den kompletten Zyklus als Basis für die Werte der Eingänge genutzt wird. Während der Zyklusabarbeitung werden die berechneten Werte für die Ausgänge in ein weiteres Prozessabbild geschrieben, welches erst nach Ende des aktuellen Bearbeitungszyklus an die Ausgangsbaugruppen übertragen wird. Somit müssen Mehrfachzuweisungen innerhalb eines Zyklus vermieden werden, da nur die letzte

[2] persistent über Funktionsaufrufe hinaus.

Zuweisung an die Ausgänge weitergegeben wird **insert PAE PAA graphic here**. Durch OBs können zusätzliche Funktionen außerhalb der zyklischen Bearbeitung realisiert werden. Beispielsweise können im Startup-OB einmalig Anweisungen beim Hochfahren der CPU ausgeführt werden.

4.2 Realisierung der Anlagentopologie

Als Grundlage für die Wegfindung muss zunächst die Anlage definiert werden. Wie bereits in Kapitel 3.1 beschrieben, wird die Anlage durch einen Graphen mit positiven Kantengewichtungen dargestellt. Für die technische Realisierung der Anlage ist unter anderem wichtig, dass die Beschreibung der Topologie erweiterbar ist, und sich der physikalische Aufbau leicht umsetzen lässt.

4.2.1 Gewählte Anlagentopologie

Im Hinblick auf die in Kapitel 2.3 definierten Implementierungsstufen und die dazu gehörigen Use-cases wurde für die Topologie der Anlage eine Matrixstruktur gewählt. Bei der Platzierung der Bearbeitungsstationen an die Außenkanten der 2x4 Wabenstruktur können die Fahrstrecken im Inneren als eine Art Überholspur für Fahrzeuge genutzt werden, die sich schneller durch die Anlage bewegen. Zudem wurden Eingangs- und Ausgangsknoten als Start und Endpunkt für die Wegfindung definiert. Als zusätzliche Rückführstrecke für Fahrzeuge nach Beendigung ihres Bearbeitungsauftrags wurde eine Rückführstrecke von dem Endzum Startknoten der Anlage hinzugefügt, welche aber in der Wegberechnung nicht berücksichtigt wird. Der Gesamtaufbau der Anlage wird in **INSERT GRAPHIC HERE** schematisch dargestellt.

4.2.2 Physikalischer Aufbau

In Kapitel 2.1 wurde dargelegt, dass die Anlage als Vorführmodell für Konzepte von Industrie 4.0 verwendet werden soll. Somit ist die Portabilität der Anlage ein wichtiger Punkt. In Zusammenarbeit mit Herrn A. Meier wurde als Basis der Anlage ein modulares System aus leichten Kartonplatten ausgewählt. Aufbauend hierauf wurden für die optische Sensorik der Fahrsteuerung die Fahrstrecken mit dunklem Isolierband aufbracht. Diese lassen sich bei Bedarf auch entfernen oder modifizieren, um Änderungen der Anlagentopologie zu simulieren. Die Kreuzungen der Fahrstrecken, die die Entscheidungspunkte für die Fahrzeugsteuerung und die Wegfindung darstellen, entsprechen den Knoten des korrespondierenden Anlagengraphen. Die Fahrstrecken, als Verbindungen zwischen einzelnen Entscheidungspunkten, entsprechen somit den Kanten des Graphen.**insert photo Anlage**

Für die Wegfindung ist es zudem wichtig die aktuelle Position eines Fahrzeugs ermitteln zu können, um diese als Basis für den Startknoten der Berechnung benutzen zu können. Aus diesem Grund wurden unterschiedliche Methoden zur Orientierung der Fahrzeuge innerhalb der Anlage untersucht. Da eine zentrale Erfassung der Fahrzeugpositionen durch zusammengeschaltete mechanische, analoge oder optische Sensoren nicht in das Konzept der Dezentralisierung von Industrie 4.0 passt, wurde ein Radio Frequency Identification (RFID)-basiertes Identifikationssystem für die Positionserfassung ausgewählt. Dieses funktioniert durch die Anbringung von RFID-Transpondern auf der Unterseite der Kartonplatten, jeweils auf Höhe eines Entscheidungsknotens. Jedes Fahrzeug besitzt einen RFID-Sensor, welcher vor dem Fahrzeug an einem Ausleger befestigt ist, um Anlagenknoten zu identifizieren, an die sich das Fahrzeug annähert. Hier hat sich die Entscheidung für Kartonplatten zur Anlagenmodellierung als Vorteil erwiesen, da die Transpondersignale auch durch den Karton auf der Oberseite der Anlagenplatten detektierbar ist, und die Transponder somit nicht auf der Fahrstrecke selbst befestigt werden müssen, was zu Behinderungen beim Fahren führen könnte.**insert photo RFID**

4.2.3 Beschreibung als Knotenliste

Sowohl für die Wegfindung als auch für die Fahrsteuerung wird eine digitale Repräsentation der Anlage benötigt. Aufgrund der einfachen Erweiterbarkeit wurde hier die Darstellungsform der Knotenliste für Graphen ausgewählt, da bei dieser Datenstruktur einfach Knoten hinzugefügt und entfernt werden können. Für einen Einzelknoten wurde die Anzahl der möglichen Kanten pro Knoten auf vier beschränkt. Dies macht eine Zuordnung der einzelnen Kanten zu den vier Himmelsrichtungen möglich. Dies wird vor allem für die Fahrsteuerung benötigt, da hier eine der Anforderungen die Möglichkeit der Ausführung von 90-Grad-Kurven ist. Da die Fahrsteuerung als getrenntes Programmmodul implementiert wurde besitzt sie eine Kopie der Knotenliste der Anlage mit den zusätzlichen, bei der Wegfindung nicht benötigten Knoten der Rückführstrecke. Ein Einzelknoten hat folgenden generellen Aufbau:

ID	Nummer des aktuellen Knoten	
Kanten	ID Nord	verbundener Knoten in Richtung Norden
	Dist. Nord	Abstand zum Knoten in Richtung Norden
	ID Ost	verbundener Knoten in Richtung Osten
	Dist. Ost	Abstand zum Knoten in Richtung Osten
	ID Süd	verbundener Knoten in Richtung Süden
	Dist. Süd	Abstand zum Knoten in Richtung Süden
	ID West	verbundener Knoten in Richtung Westen
	Dist. West	Abstand zum Knoten in Richtung Westen

Da in STEP7 Speicher nicht dynamisch alloziert werden kann, hat jeder Knoten immer Speicherplatz für die Daten von allen vier möglichen Kanten, auch wenn er in der realen Anlage mit weniger als vier Knoten verbunden ist. Da die ID „0“ für den Startknoten reserviert wurde, werden Verbindungsrichtungen durch eine „-1“ im ID-Feld als nicht verbunden gekennzeichnet. Hierdurch wird auch der zugehörige Abstand ignoriert.
Die Datenstruktur der Knotenliste selbst besteht aus einem einfachen Array mit Einzelknoten als Arrayelementen. Dieses Array wird initialisiert mit einer Konstanten, welche die Gesamtanzahl der möglichen Anlagenknoten enthält. Zusätzlich zu dem Knotenarray besteht der Listendatentyp noch aus der Anzahl der wirklich im Array enthaltenen Knoten. Diese wurde aus Gründen der Konsistenz hinzugefügt , da andere selbst definierte Listentypen im Projekt mit variablen Anzahlen von Arrayelementen arbeiten, was bei der Anlagentopologie nur selten der Fall ist. Dies könnte in diesem Falle genutzt werden wenn zu einem späteren Zeitpunkt weitere Knoten zur Anlage hinzugefügt werden. Hier muss aber ein Neustart des FTF durchgeführt werden, da sonst die in Kapitel 3.6 besprochene Heuristik-Berechnung für die neu hinzugefügten Knoten nicht existiert und somit die Knoten nicht für die Wegberechnung verwendet werden können.**Insert Graphic Datatype TIA**
Abschließend sei noch erwähnt das die in Abschnitt 4.2.2 erwähnten RFID-Werte der Knoten nicht für die Wegfindung relevant sind. Das Einlesen und die Zuordnung von RFIDs zu den entsprechenden Knoten wird komplett in der Fahrzeugsteuerungsschicht erledigt, welche der Wegfindungsschicht dann nur die interne ID weitergibt.

4.2.4 Beschreibung als Adjazenzmatrix

Unter dem Aspekt der Erweiterbarkeit ist die Listendarstellung des Graphen ideal, da Knoten unabhängig von ihrer Position innerhalb der Anlage einfach am Ende der Liste angehängt werden können. Solange der neue Knoten gültig ist, kann der erweiterte Anlagengraph verwendet werden. Für die eigentliche Wegfindung

ist eine solche Liste jedoch ungünstig, um schnell den Abstand eines Knotens zu einem beliebigen anderen Knoten zu prüfen. Bei einer Liste müsste im schlechtesten Fall bei Überprüfung des zuletzt hinzugefügten Knotens das komplette Array durchlaufen werden. Aus diesem Grund wird beim Hochfahren der CPU, vor der Berechnung der Heuristik mittels Dijkstra, die Knotenliste geparst und eine Adjazenzmatrix der Anlage generiert. Hier kann in konstanter Zeit[3] der Abstand zwischen zwei Knoten x und y ermittelt werden, indem einfach der in der Matrix unter den Indizes (x, y) hinterlegte Wert abgerufen wird. Dies beschleunigt vor allem die Laufzeit des im zyklischen Betrieb ausgeführten A*-Algorithmus. Für die Berechnung der Heuristiken ist dies weniger von Bedeutung, da hier keine Zeitbeschränkungen vorliegen und nur der weniger kritische Aspekt der kürzeren Hochfahrzeit der SPS beeinflusst wird.**insert graphic here**

4.3 Implementierung der Algorithmen

Das gesamte Programm wurde in einzelne funktionale Schichten aufteilt, um den Industrie 4.0 Aspekt der Modularisierung zu entsprechen. Somit macht es keinen funktionalen Unterschied, ob die Wegfindungsschicht tatsächlich auf derselben Steuerung läuft wie die Fahrzeugsteuerungsschicht. Als Schnittstelle dienen hier sowohl bei der Wegfindungs- als auch bei der Fahrzeugsteuerungsschicht Datenbausteine, in welche die zu übertragenden Ergebnisse geschrieben werden. Diese Daten werden durch die in Kapitel 5.1 näher beschriebene Kommunikationsschicht übertragen. Dies hat den Vorteil das unterschiedliche Arten der Kommunikation einfach ausgetauscht werden können, ohne die anderen Schichten zu beeinflussen.**insert schicht graphic here**.
Die in Kapitel 3 beschriebenen Algorithmen wurden beide auf ähnliche Weise in STEP7 implementiert. Für die A* und Dijkstra-Algorithmen wurden jeweils geeignete Datentypen definiert, welche an die Anforderungen der Algorithmen angepasst wurden. Die Implementierung verwendet hier in beiden Fällen eine Priority-Queue auf Basis eines Arrays, um bei der Berechnung den jeweils nächsten „'besten" Knoten zu identifizieren. Das Array als Grundlage der Prioritätsschlange wurde hier gewählt, da sich die Realisierung in STEP7 einfacher gestaltet als beispielsweise die eines Heaps. Dafür müssen hier Abstriche bei der Sortiergeschwindigkeit gemacht werden.

4.3.1 Berechnungsdatentypen

Für die Berechnung der Algorithmen wurden Datentypen erstellt, die alle für den Algorithmus benötigten Informationen enthalten. Die Datentypen beschrei-

[3] Aufwand $\mathcal{O}(1)$

ben die Knoten einzeln und werden zur Berechnung in eine Priority-Queue eingefügt. Als Ergebnis der Berechnung wird eine geordnete Liste der Einzelknoten ausgegeben. Die Datentypen enthalten folgende Grunddaten:

NodeID	ID des betrachteten Knotens
Dist. to Start	Bisher zurückgelegter Weg
Expected Dist.	Vorausichtl. Gesamtpfadlänge durch diesen Knoten (nur A*)
ParentID	Vorgängerknoten

Der bisher zurückgelegte Weg gibt bei der Dijkstra-Implementierung Aussagen darüber, wie weit ein Knoten von einem Startknoten entfernt ist. Wird dieser Startknoten später als Ziel gewählt, so kann der Abstand als Heuristik für A* verwendet werden[4]. Da jeweils der kürzeste Weg gesucht ist, wird für diese Datenkomponente sowie auch für den voraussichtlichen Gesamtweg der Initialwert auf den Maximalwert des verwendeten Datentyps gesetzt, in diesem Fall 65535 für 16-Bit Integer.
Insert graphic node here
Die voraussichtliche Pfadlänge bei dem Datentyp für den A*-Algorithmus setzt sich zusammen aus dem bisher zurückgelegten Weg und dem geschätzten Wert für den restlichen Weg zum Ziel. Bei Verwendung einer exakten Heuristik ist dieser Wert für alle Knoten auf dem kürzesten Pfad immer konstant, solange sich die Anlage seit Berechnung der Dijkstra-Heuristik nicht verändert hat.
Der Zeiger zum Vorgängerknoten wurde hier als einfache Zahl implementiert, da das Pointer-Konstrukt in STEP7 etwas anders funktioniert als beispielsweise in der Programmiersprache C. Dies hat den Nachteil das bei der Zusammenstellung der Route[5] anhand der Knotenliste etwas mehr Zeit verwendet werden muss, da die Liste jeweils nach dem Vorgängerknoten durchsucht wird.

4.3.2 Priority-Queue

Da in STEP7 keine Container-Datentypen existieren, die mit unterschiedlichen Elementen zurechtkommen, wurden für beide Algorithmen getrennte Datentypen für die Priority-Queue definiert. Diese existieren analog zu der in Abschnitt 4.2.3 beschriebenen Anlagenknotenliste aus einem Array der Knotenelemente und der Anzahl der beinhalteten Elemente. Dies wurde aus Gründen der Effizienz auf diese Art realisiert, da der große Vorteil des A*-Algorithmus vor allem darin liegt, dass nur ein kleiner Teil der Knoten der Anlage betrachtet werden muss. Da das Array aber nicht dynamisch mit der benötigten Größe alloziert werden kann, sondern immer den schlechtesten Fall berücksichtigen muss, wurde

[4] Bei gerichteten Graphen muss zusätzlich die Gegenrichtung betrachtet werden.
[5] beschrieben in Abschnitt 4.3.5.

zur Initialisierung des Arrays die Gesamtanzahl der Anlagenknoten als Konstante verwendet. Damit somit bei den Priority-Queue-Operationen nicht die leeren Restplätze des Arrays mit berücksichtigt werden, wurde die Anzahl der Knoten als Datenkomponente mit hinzugefügt. Besteht die Anlage beispielsweise aus 40 Knoten aber es befinden sich nur 6 echte Knoten in der Prioritätsschlange, so laufen alle Schleifen nur über die Elemente null bis fünf.
Zur Verwendung der Priority-Queue wurden für beide Listendatentypen Zugriffs- und Verwaltungsfunktionen implementiert. STEP7 besitzt zwar die Möglichkeit Funktionen mittels eines sogenannten VARIANT-Pointers so zu implementieren, dass sie analog zum Konzept der Überladung bei Objektorientierter Programmierung mit unterschiedlichen Eingangsdatentypen zurechtkommen[13], dies ist in diesem Fall aber nicht möglich, da nicht effizient[6] auf die Einzelelemente des Datentyps zugegriffen werden kann. Aus diesem Grund wurden die folgenden Operationen für jeden der beiden Datentypen implementiert:

- **Insert**: Einfügen eines Knotens in die Priority-Queue.
- **Pop**: Ausgabe und Entfernung des nach der Prioritätsbedingung „besten“ Knotens aus der Priority-Queue.
- **Sort**: Wiederherstellung der Prioritätsschlangeneigenschaften durch Sortierung.
- **Contains**: Überprüfung, ob Knoten bereits in der Schlange enthalten ist.
- **UpdateNode**: Modifizierung eines Knotens in der Priority-Queue.

Die Operationen „Insert“, „Pop“ und „UpdateNode“ beinhalten alle eine abschließende Wiederherstellung der Prioritätsbedingung mittels der in **INSERT CODE REFERENCE HERE** dargestelten „Sort“-Funktion.
Um die Zugriffsoperationen effizienter zu machen, wurden die Prioritätsschlangen so implementiert, dass das beste Element der letzte gültige Knoten im Array an dem Index „$Anzahl - 1$“ ist. Dies Vereinfacht das Entfernen des besten Knotens, da nach der Ausgabe nur die Anzahl um eins dekrementiert wird und nicht jeder Knoten in Richtung des Arrayanfangs nachgerückt werden muss, wie es der Fall wäre wenn das beste Element am Anfang stehen würde.

4.3.3 Dijkstra-Algorithmus und Heuristik-Tabelle

Die eigentliche Implementierung des Dijkstra-Algorithmus ist recht einfach. Zunächst muss die Prioritätsschlange mit allen Knoten der Anlage gefüllt[7]

[6] es können nicht Elemente sondern nur Datenbereiche adressiert werden.
[7] Gruppen II und III aus Abschnitt 3.4.1 werden gemeinsam in der Prioritätsschlange gehalten.

und die Distanz des Startknotens auf null gesetzt werden. Da hier keine Route gesucht wird sondern nur ein Abstand zum Startknoten, wird der Algorithmus nicht beendet wenn ein bestimmter Zielknoten erreicht wurde, sondern erst wenn alle Knoten betrachtet worden sind. Bereits betrachtete Knoten werden gemäß des in Abschnitt 3.4.1 beschriebenen Prinzips aus der Priority-Queue in eine geordnete Knotenliste übertragen. Dieser, Heuristikliste genannte, Datentyp enthält die Knotenelemente aufsteigend sortiert nach Abstand zum Startknoten.
Der gesamte Dijkstra-Algorithmus wird in einem FC mit dem Namen „FC_buildHeuristicList“ berechnet **Insert graphic FC here**. Dieser Baustein wird für jeden Knoten der Anlage aufgerufen, der als Endzielknoten[8] markiert ist. Wie bereits erwähnt wird dieser Baustein in einer Schleife im „OB100 Startup“ aufgerufen. Die resultierenden Heuristiklisten werden in einem Datentyp namens Heuristiktabelle in deren Arraykomponente gespeichert und im zyklischen Betrieb von dem A*-Algorithmus verwendet.

Notiz: Bausteine näher beschreiben? mit Auflistung und Erklärung aller Schnittstellen?

4.3.4 A*-Algorithmus

Die Implementierung des A*-Algorithmus ähnelt der des Dijkstra-Algorithmus. Auch hier muss vor Beginn der Berechnung zunächst die Priority-Queue vorbereitet werden. Da der A*-Baustein im Gegensatz zu dem Dijkstra-Baustein mehr als einmal aufgerufen wird, bedeutet dies, dass die Prioritätsschlange zunächst in den Grundzustand versetzt werden muss, indem alle Knoten aus vorherigen Berechnungen entfernt werden. Zudem muss die Liste der geschlossenen Knoten, die in Abschnitt 3.5.1 erwähnt wurde, komplett geleert werden, damit kein Knoten bei der Routenberechnung fälschlicherweise als bereits betrachtet angesehen wird.
Die Berechnung selbst startet nun mit dem Hinzufügen des Startknotens zur Priority-Queue, welche die Liste der offenen Knoten[9] darstellt. Es wird nun immer der beste Knoten aus der Priority-Queue herausgenommen und als geschlossen markiert. Im Anschluss wird die Adjazenzmatrix nach neuen, vorher nicht erreichbaren Knoten durchsucht. Falls diese noch nicht als geschlossen markiert wurden wird geprüft, ob die Verbindungskante zu dem neuen Knoten in einer speziellen Matrix für permanente Blockaden als blockiert gekennzeichnet wurde. Ist dies nicht der Fall so wird der neue Knoten zu der Liste der offenen Knoten hinzugefügt. Der fertig betrachtete geschlossene Knoten wird dann in die statische Routenliste eingefügt, aus der nach dem Ende der Berechnung die Route

[8] vergleiche Abschnitt 3.6.
[9] Open-List

generiert wird. Wurde der Zielknoten erreicht oder befinden sich keine Knoten mehr in der Priority-Queue, so beendet sich der Algorithmus. Die Routenliste wird in die Ausgangsvariable kopiert und der Baustein meldet das Berechnungsende über einen Bitmerker an die übergeordnete Wegfindungsschicht.
Die Erkennung von permanenten Blockaden wurde durch eine Matrix realisiert, da hier, wie auch bei der Adjazenzmatrix, in konstanter Zeit geprüft werden kann ob auf dem betrachteten Streckenabschnitt Behinderungen vorliegen. Diese permanenten Blockaden liegen über längere Zeitintervalle vor und können deshalb bereits zu Beginn der Routenberechnung berücksichtigt werden. Dies ist bei temporären Blockaden durch andere Fahrzeuge nicht der Fall, weshalb zur Erkennung dieser Art von Behinderungen ein anderer Mechanismus zur Anwendung kommt, der in Abschnitt 5.2 näher beschrieben wird.

4.3.5 Generation der Route

Der A*-Algorithmus liefert als Ergebnis eine Liste mit allen geschlossenen Knoten. Aus dieser Liste muss nach Beendigung des Algorithmus eine Route generiert werden, welche in für die Fahrzeugsteuerung verständlicher Form vorliegt. Da die Fahrzeugsteuerungsschicht ebenso wie die Wegfindung über eine Knotenliste der Anlagentopologie verfügt[10], genügt es, nur eine Liste der KnotenIDs des errechneten kürzesten Pfades zu übertragen.
Es besteht jedoch das Problem, dass in der Liste der geschlossenen Knoten, auch solche vorkommen können, die nicht Teil des kürzesten Weges sind. Dies ist vor allem dann der Fall, wenn sich die Anlage zwischenzeitlich geändert hat. Würden hier nur die IDs aller Knoten der Liste übertragen, würde dies zu ungültigen Routen für die Fahrteuerung führen, obwohl der Algorithmus einen validen Weg ermittelt hat. Dies ist der Grund, warum die in Abschnitt 4.3.1 vorgestellten Datentypen über die Komponente „ParentID“ verfügen. Diese hilft bei der Rückverfolgung des Pfades vom Ziel aus bis zum Startknoten. Um somit eine Route zu generieren, muss zunächst der Zielknoten gefunden werden und dann anhand der ParentIDs der Pfad zum Start einzeln nachverfolgt werden. Der Zielknoten selbst kann einfach gefunden werden, da er mit großer Wahrscheinlichkeit der letzte geschlossene Knoten der Knotenliste ist, da das Erreichen des Ziels eine der Abbruchbedingungen des Algorithmus ist. Die andere Bedingung ist, das keine weiteren Knoten in der Liste der offenen Knoten vorhanden sind. In diesem Fall existiert aber kein Weg vom Start- zum Zielknoten und die Generation der Route ist hinfällig.
Da die Reihenfolge beim Parsen der Knoten jedoch invers zu der für die Fahrzeugsteuerung benötigten Reihenfolge der Route ist, werden die zum kürzes-

[10] wobei für die Fahrzeugsteuerung nur die Verbindungen eines Knotens und nicht die Kantengewichtungen von Bedeutung sind.

ten Weg gehörenden Knoten temporär in einer Liste abgespeichert, aus der dann durch Umkehrung die eigentliche Route erzeugt wird. Nachdem nun der Zielknoten gefunden wurde, muss mittels der ParentID der vorletzte Knoten der Route gefunden werden. Um nicht auf der Suche nach einem Knoten bei jedem Suchdurchlauf die komplette Knotenliste durchsuchen zu müssen, kann davon ausgegangen werden, dass sich der Vorgänger eines Pfadknotens an einer Stelle in der Liste der geschlossenen Knoten befindet, die vor dem Index des Pfadknotens liegt. Dies lässt sich gemäß Abschnitt 3.4.1 aus der Bedingung für geschlossene Knoten folgern. Bei einer unterschätzenden Heuristik für den A*-Algorithmus wurde für jeden geschlossenen der kürzeste Pfad ausgehend vom Startknoten bereits gefunden. Es ist zudem sinnvoll in absteigender Reihenfolge nach dem Vorgängerknoten zu suchen, da dieser wahrscheinlicher in der Nähe des Pfadknotens selbst zu finden ist als beim Startknoten. Somit kann der Aufwand der Routengeneration auf $\mathcal{O}(n) \leq \mathcal{O}(Routengeneration) \ll \mathcal{O}(n^2)$ beschränkt werden.

4.4 Einhaltung der Echtzeitbedingung

Wenn die Wegfindungsschicht auf der gleichen SPS wie die Fahrzeugsteuerungsschicht implementiert wird, so muss die Wegberechnung bestimmte zeitliche Anforderungen erfüllen, damit die Fahrzeugsteuerung nicht beeinträchtigt wird. Eine Steuerung arbeitet im zyklischen Betrieb alle OBs nacheinander ab, die als zyklisch definiert wurden. Durch die Modularität des Anwenderprogramms besitzt jede Schicht ihren eigenen zyklischen OB, da nur so eine wirkliche Trennung der Schichten möglich ist. Dies bedeutet, dass die Berechnung eines Weges die CPU nicht so lange in Anspruch nehmen darf, dass in dieser Zeit ein überfahrener RFID-Knoten nicht eingelesen und somit die bevorstehende Kreuzung nicht erkannt wird. Dies führt spätestens beim nächsten erreichten Knoten zu der Erkennung eines Fehlers und das Fahrzeug geht in einen permanenten Fehlerzustand. Ebenso kann es bei zu langer CPU-Belegung vorkommen, dass das Fahrzeug die Spur verliert, dies erst bemerkt wenn es komplett von der Fahrstrecke abgekommen ist und dadurch nicht allein zurück auf die Strecke findet. Aus diesem Grund ist es sehr wichtig, die Wegfindungsschicht so anzupassen, dass zwischen zwei Ausführungen der Fahrzeugsteuerung höchsten 10ms liegen. Da auch die Kommunikationsschichten zyklische OBs besitzen, die der Reihenfolge nach abgearbeitet werden müssen, wurde festgelegt, das die Wegberechnung maximal 5ms dauern darf. Dies hat einige Konsequenzen für die Implementierung der Wegfindungsalgorithmen.

4.4.1 Ausführung bei Systemstart

Wie bereits mehrfach erwähnt wird die Heuristiktabelle beim Hochfahren der CPU generiert. Eigentlich ist es nicht notwendig für den A*-Algorithmus, dass bereits ein Wert für die Heuristik existiert. Es könnte auch im zyklischen Betrieb ein Wert für die Abschätzungsfunktion $f(n)$ aus Abschnitt 3.5.2 berechnet werden, beispielsweise mittels einer Funktion für die Manhattan-Distanz, welche den Abstand zweier Knoten anhand der Differenz derer Koordinaten ermittelt. Dies würde den Speicherverbrauch erheblich reduzieren, da keine großen Tabellen gespeichert werden müssten, sondern nur die jeweiligen Koordinaten der Knotenpunkte.
Die Berechnung einer Funktion $f(n)$ ist aber zeitintensiver als das Nachschlagen von Werten in einer Tabelle. Da der Speicherbedarf für die Tabelle der realisierten Anlage mit 25 Knoten und 10 Endzielknoten nur insgesamt 2300 Byte beträgt, fällt dies selbst bei der kleinsten Steuerung vom Typ S7-1214C mit 100kB Arbeitsspeicher kaum ins Gewicht. Zugleich kann die Laufzeit des A*-Algorithmus im zyklischen Betrieb deutlich reduziert werden, da die Tabellenheuristik mittels Dijkstra, im Gegensatz zur Manhatten-Distanz, eine exakte Heuristik liefert, die zusätzlich zur eingesparten Rechenzeit der Tabellenaufrufe noch die Anzahl der betrachteten Knoten minimiert.
Da die Berechnung des Dijkstra-Algorithmus aber unter anderem auch wegen der gewählten Implementierung mit arraybasierten Priority-Queues zeitintensiv ist [14] und einige Sekunden dauern kann, muss die Tabellengenerierung zu einem Zeitpunkt geschehen, an dem die Echtzeitbedingung der Fahrzeugsteuerungsschicht nicht greift. Aus diesem Grund wurde der Hochfahrvorgang der CPU gewählt, um die notwendigen Initialisierungen der Heuristiktabelle durchzuführen, da hier weder die Fahrzeugsteuerung noch der Watchdogtimer, welcher den zyklischen Betrieb standardmäßig auf 50ms beschränkt, aktiv ist. Zusätzlich wird zu diesem Zeitpunkt noch die Adjazenzmatrix aus der Anlagentopologie generiert, da auch der Parsevorgang der Anlagenknoten bei größeren Anlagen etwas mehr Zeit in Anspruch nehmen kann. Zudem kann somit die Adjazenzmatrix für die schnellere Berechnung des Dijkstra-Algorithmus verwendet werden.

4.4.2 Zyklische Ausführung

Trotz der Auslagerung der Heuristikberechnung auf die Hochlaufphase der CPU kann bei langen Routen nicht garantiert werden, das die als Ziel gesetzten 5ms für die Berechnungszeit eingehalten werden. Vor allem wenn sich die Anlage seit dem Hochfahren des Fahrzeugs stark verändert hat und somit die Heuristiktabelle keine exakten Heuristikwerte mehr liefert, müssen unter Umständen zusätzliche Knoten betrachtet werden, die nicht auf dem kürzesten Pfad vom

Start- zum Zielknoten liegen. Der Aufwand für die Berechnung verschiebt sich also immer mehr nach $\mathcal{O}(|\mathcal{V}|^2)$ wie in Abschnitt 3.5.4 erläutert wurde.

Aus diesem Grund wurde der A*-Algorithmus so implementiert, das der Zeitaufwand pro Zyklus größtenteils konstant bleibt, unabhängig von der Länge der Berechneten Route. Dies wurde erreicht indem die Schleife, welche läuft, solange das Ziel noch nicht erreicht ist, aufgetrennt und durch eine konditionale IF-Abfrage ersetzt wurde. Nach der Schleifenauftrennung wird immer nur ein Knoten pro Zyklus untersucht und gegebenenfalls seine Verbindungsknoten zur Open-List hinzugefügt, weitere Knoten werden erst in den darauf folgenden Zyklen expandiert. Dies hat zur Folge, dass zwar die Bearbeitungszeit pro Zyklus sehr kurz gehalten wird, im implementierten Beispiel wird eine Zykluszeit der Wegfindungsschicht kleiner 1ms erreicht, die Zeit bis zur Berechnung einer kompletten Route jedoch deutlich länger wird. Dies ist der Fall, da zwischen zwei Bearbeitungszyklen immer die kompletten OBs der anderen Schichten abgearbeitet werden. Dieses Problem kann aber zunächst vernachlässigt werden und wird im Abschnitt 5.3.2 näher behandelt.
Ein zweites Problem, das bei der Aufteilung des Algorithmus auf mehrere Zyklen auftritt, ist die Inkonsistenz von Routen. Während der Berechnung des Algorithmus darf die in Abschnitt 4.3.4 erwähnte extern gespeicherte Routenliste nicht als gültige Route missverstanden werden, da hierdurch das Fahrzeug eine möglicherweise falsche Route nehmen würde, die nicht zum Ziel führt. Somit darf der Baustein nur finale Routen ausgeben, und temporäre Routen nur intern speichern. Da der Baustein jedoch in mehreren Zyklen hintereinander aufgerufen wird und zwischen zwei Aufrufen die bereits berechneten Daten nicht verloren gehen dürfen, muss er vom Typ FB sein, da nur diese Bausteine einen zugeordneten Instanz-Datenbaustein besitzen, in dem Werte über den Funktionsaufruf hinaus zwischengespeichert werden können.
Durch die Kombination dieser beiden Maßnahmen, kann erreicht werden, dass die CPU-Belegung durch die Wegfindungsschicht im zyklischen Betrieb, quasi konstant[11] auf unter 1ms gehalten werden kann.

4.5 Steuerung des Bearbeitungsablaufs

Die Wegfindung innerhalb der Anlage besteht nicht allein aus der Berechnung von kürzesten Pfaden zwischen zwei Anlagenknoten. Sie muss auch wissen, welcher Bearbeitungsschritt als nächstes ausgeführt werden soll und welcher Anlagenknoten die korrespondierende Funktionalität besitzt. Zudem soll es möglich sein, dass mehrere Bearbeitungsstationen die gleiche Funktionalität bereitstellen.

[11] immer noch abhängig von der Kantenanzahl des aktuell betrachteten Knotens.

In diesem Fall muss eine geeignete Station ausgewählt und der Weg zu dem entsprechenden Knoten berechnet werden können.

4.5.1 Bearbeitungsreihenfolge

Im Abschnitt 2.2.2 wurde definiert, dass das Fahrzeug selbst die Ablaufreihenfolge der Bearbeitungsschritte kennt. Diese Anforderung wurde durch die Implementierung des Datentyps „Assignment“ realisiert. Dieser Datentyp stellt eine Art Rezept dar, in dem gespeichert ist, welche Funktionalitäten ein bestimmtes Werkstück benötigt[12]. Er ist zusammengesetzt aus der Anzahl der benötigten Funktionalitäten sowie einer Liste der zugehörigen FunktionalitätsIDs. Hierfür muss jedoch zunächst definiert sein, welche Funktionalitäten in der Anlage zur Verfügung gestellt werden und wie die Knoten lauten, die diese Funktionalität bereitstellen. Für diese Aufgabe wurde ein Datentyp erstellt, welcher wie folgt aufgebaut ist:

FunctionID	Kennziffer der bereitgestellten Funktionalität.
Nr. of Stations	Anzahl Stationen, die diese Funktionalität bereitstellen.
Cluster	KnotenIDs der Stationen, die diese Funktionalität besitzen.

Dieses Funktionscluster ermöglicht es nun, Rezepte unabhängig von spezifischen Bearbeitungsstationen zu formulieren. Die Bearbeitungsreihenfolge eines Werkstücks kann somit ausgedrückt werden durch eine Liste aufeinanderfolgender Funktionalitäten, die ein bestimmtes Werkstück benötigt. Im Verlauf der Abarbeitung des Rezepts wählt das Fahrzeug selbst eine der im Cluster hinterlegten Stationen zur benötigten Funktionalität aus. Das Rezept muss der Wegfindungsschicht während des gesamten Bearbeitungsvorgangs zur Verfügung stehen, um die Auswahl des jeweils nächsten Routenziels zu ermöglichen. Daraus folgt, das Rezepte nach der Zuweisung zu einem Werkstück bis zum Abarbeitungsende gespeichert werden müssen.
Zur Erfassung der Daten von Fahrzeugen, werden die folgenden Informationen gespeichert:

- ID des Fahrzeugs, zur Unterscheidung unterschiedlicher Fahrzeuge.
- ID des zuletzt besuchten Knotens innerhalb der Anlage.
- ID des nächsten Knotens.
- ID des Zielknotens der Aktuellen Route.
- Indexnummer des derzeitigen Bearbeitungsschritts innerhalb des zugewiesenen Rezepts.

[12] Anlageneinn- und Ausgänge sind auch Funktionalitäten, die erfasst werden.

- Das zugewiesene Rezept.

Für die in Abschnitt 5.2.1 beschriebene Dynamische Wegfindung ist es notwendig, diese Art von Fahrzeugdaten für alle Fahrzeuge in der Anlage zu speichern. Die ersten drei Punkte sind für alle erfassten Fahrzeuge wichtig, die restlichen Informationen müssen zum derzeitigen Implementierungsstand nur vom eigenen Fahrzeug erfasst werden. Mittels der ID des letzten und des nächsten Knotens kann die Graphenkante identifiziert werden, auf der sich das Fahrzeug momentan befindet. Durch Vergleich des zuletzt besuchten Knotens mit dem Zielknoten der Route wird das Ende der Teilroute erkannt. Auf welchem Weg die benötigten Informationen gesammelt werden, wird im Kapitel 5.1 näher betrachtet.**insert graphic of vehiclecontrolblock with assignment here**
Die Zuweisung von Rezepten erfolgt für jedes Fahrzeug beim Eintritt in die Anlage. Der Anlagenknoten „0“ stellt in der realisierten Modellanlage den Startknoten dar. Bei Erkennung des Startknotens wird ein Rezept aus der verfügbaren Rezeptliste ausgewählt und dem Fahrzeug zugewiesen. Nach einer kurzen Wartezeit startet die Wegfindungsschicht die Berechnung der Route zur ersten Funktionalität des zugewiesenen Rezepts. Beim Verlassen der Anlage deaktiviert sich die Wegfindung und es wird eine statische Route über die Rückführstrecke zum Anlageneingang zurück gesendet. Beim Erreichen des Eingangsknotens wird erneut ein Rezept zugewiesen. Dies ermöglicht eine endlose Wiederholung des Bearbeitungsvorgangs mit unterschiedlichen Rezepten bis zum Abbruch durch den Benutzer.
Ggf für Fazit oder hier: Eine geplante Erweiterung der Anlage sieht vor, dass die Zuweisung von Rezepten durch ein zentrales Steuersystem außerhalb der Anlage abgewickelt wird. Diese Funktion wurde jedoch noch nicht implementiert, weshalb die Zuweisung innerhalb der Wegfindungsschicht stattfindet.

4.5.2 Zuweisung einer Bearbeitungsstation

Existiert pro Funktionalität nur eine Station, so ist die Zuweisung trivial. Existieren aber mehrere Bearbeitungsstation innerhalb eines Funktionalitätsclusters, so muss eine Station ausgewählt werden, welche als Ziel für die nächste Routenberechnung dient. Ein Problem, dass hierbei auftreten kann, ist die ungleichmäßige Auslastung einzelner Bearbeitungsstationen innerhalb eines Clusters. Abbildung **Insert graphic auslastungsbeispiel here** zeigt zwei Stationen mit der gleichen Funktion, wobei sich eine Station näher am Anlageneingang befindet als die zweite. Würden Bearbeitungsstationen nur nach dem Kriterium des kürzesten Weges und des Belegungszustands ausgewählt, so würden alle Fahrzeuge, welche die Anlage betreten, in den meisten Fällen Station 1 anfahren, Station 2 würde nur dann ausgewählt wenn Station 1 belegt ist oder aufgrund von Streckenblo-

ckaden der Weg zu Station 2 kürzer ist. Dies würde zu einer verhältnismäßig höheren Auslastung von Bearbeitungsstation 1 führen. Dies ist im industriellen Umfeld jedoch nicht erwünscht, da Anlagenteile meist gleichzeitig modernisiert oder erneuert werden und durch die höhere Auslastung einer Station das Ausfallrisiko dieser Komponente steigt.
Aus diesem Grund wurde entschieden, die Anlagen zufällig an eine der verfügbaren Bearbeitungsstationen des Clusters zu verteilen. Der derzeitige Belegungsstand wurde hierbei vernachlässigt, da die Bearbeitungsdauer einer Station im Modell verglichen mit der Fahrtdauer eines Fahrzeugs sehr gering ist. Somit is eine bei der Berechnung belegte Bearbeitungsstation bis zum tatsächlichen Erreichen der Station wahrscheinlich frei. Es ergab sich jedoch das Problem, dass unter normalen Umständen Anlagensteuerung exakt arbeiten, da zufällige Ergebnisse innerhalb der Produktion unerwünscht sind. Dies bedeutet das eine SPS keinen eingebauten Pseudozufallsgenerator zur Verfügung stellt und dieser somit implementiert werden muss. Zur Generierung von Zufallszahlen wurde hier die Prozessorzeit verwendet, welche durch eine Modulo-Operation mit der Anzahl der zur Auswahl stehenden Bearbeitungsstationen eine für diese Bedürfnisse ausreichende Gleichverteilung der Stationen erreicht.

4.5.3 Simulation der Bearbeitungszeit

Hat ein Fahrzeug die Bearbeitungsstation am Ende der zugewiesenen Teilroute erreicht, ist es aus Simulationssicht sinnvoll, zunächst ein bestimmtes Zeitintervall zu warten, bevor die nächste Teilroute ausgeführt wird. Hierdurch wird der Bearbeitungsvorgang innerhalb einer Bearbeitungsstation simuliert. Zur Implementierung dieser Funktion wird innerhalb der Wegfindungsschicht bei Erkennen des erreichten Endes einer Teilroute ein Hardwaretimer gestartet, der nach drei Sekunden einen Impuls von einer Periode erzeugt. Dieser Impuls wird verwendet, um die Ausführung der nächsten Teilroute zu starten. Da die Zuweisung von Bearbeitungsstationen wie in Abschnitt 4.5.2 beschrieben nicht von dem Belegungsstatus der nächsten Zielstation abhängt, kann die Berechnung der nächsten Teilroute bereits während der Simulation des Bearbeitungszustands durchgeführt werden. Durch gerichtete Verbindungen an den Ausgängen der Maschinenplätze ist zudem gewährleistet, das auf der ersten Teilstrecke kein Fahrzeug entgegen kommen kann.

4.6 Beispiel für eine einfache Routenberechnung

Das nachfolgende Beispiel zeigt graphisch die Abarbeitung eines einfachen Rezepts mit der Funktionalitätsreihenfolge $(2, 1)$. Die Funktionalität 2 wird im vorliegenden Beispiel von den Knoten X und Y bereitgestellt, bei der Funktion 1 han-

delt es sich um den Ausgang an Knoten 24. Die Funktionalität 0 ist immer den Eingängen zugeordnet, Funktionalität 1 beschreibt analog immer die Ausgänge der Anlage. Es wird angenommen das während der Abarbeitung des Rezepts weder permanente noch temporäre Blockaden auftreten:
insert multiple graphics showcasing pathfinding without dynamic reactions here.

5

Besonderheiten der Dynamischen Wegfindung

5.1 Kommunikation

Eine Grundvoraussetzung für jegliche Art von Dynamik ist das Wissen über die eigene Umgebung. Es ist nicht möglich, dynamisch auf Ereignisse zu reagieren, wenn Daten über die Umgebung seit der initialen Wegberechnung nicht aktualisiert wurden. Aus diesem Grund ist Kommunikation als Mittel der Informationsbeschaffung essentiell. Da das Anwenderprogramm modularisiert und in Schichten aufgeteilt wurde, ist ein Datenaustausch unabdingbar.
Da das Projekt von zwei Personen bearbeitet wurde und die Kommunikation zwischen beiden Teilen separat entwickelt wurde, kann die Kommunikationsschicht in zwei getrennte Unterschichten aufgeteilt werden. Jede dieser Teilschichten regelt die Schnittstelle zu der zugehörigen Schicht. Aus diesem Grund wird in folgenden Abschnitt vor allem die Kommunikation aus Sicht der Wegfindung beschrieben. Beide Unterschichten wurden aber aufeinander abgestimmt und sind im Kern identisch und haben nur unterschiedliche Verwendungsarten.
Wie bereits erwähnt, erfüllt die Kommunikationsschicht vor allem zwei Anforderungen. Die erste Anforderung ist der Datenaustausch zwischen den Schichten des Anwenderprogramms eines einzelnen FTF. Da in der vorliegenden Realisierung der Anlage sowohl die Wegfindung als auch die Fahrzeugsteuerung auf der selben Steuerung ausgeführt werden, wurde diese Kommunikationsart als „Interne Kommunikation“ bezeichnet. Im Gegensatz hierzu wurde der Austausch von Informationen zwischen anderen Fahrzeugen „Externe Kommunikation“ genannt. Es ist grundsätzlich möglich, beide Arten der Kommunikation gemeinsam zu implementieren. Da aber für den Datenaustausch innerhalb der SPS das Netz nicht unnötig belastet werden muss, wurde entschieden die beiden Arten

von Kommunikation aufzutrennen. Dies hat den weiteren Vorteil das die Daten schneller für die jeweils andere Schicht verfügbar sind, als dies beispielsweise über eine drahtlose Verbindung der Fall wäre.**insert Schichten graphic here**

5.1.1 Schnittstelle zur Kommunikationsschicht

Beide Arten der Kommunikation besitzen die selbe strukturellen Anbindung an die zugehörige Programmschicht. Für die ein- und ausgehende Kommunikation existiert jeweils ein Schnittstellen-DB. Beide DBs besitzen den folgenden grundsätzlichen Aufbau:

- Speicherplatz für Daten zur internen Kommunikation.
- Speicherplatz für Daten zur externen Kommunikation.
- Signalisierungsbits zum Erkennen oder Anstoßen der Kommunikation.

Am Beispiel der Wegfindungsschicht wird besteht der Schnittstellen-DB für die ausgehende Kommunikation aus einem Datentyp für die in Abschnitt 4.3.5 beschriebenen Routeninformationen für den internen Datenaustausch, sowie einem Byte-Array konstanter Größe für die externe Kommunikation mit anderen Fahrzeugen. Da die Wegfindung als einzigen Partner einer Ausgehenden Verbindung jedoch die korrespondierende Fahrzeugsteuerung besitzt, wird das Byte-Array bei der Implementierung aller Schichten auf der gleichen Steuerung nicht benötigt. Das vorhandene Signalisierungsbit gibt eine Sendeanforderung für die Route aus der Wegfindungsschicht an die Kommunikationsunterschicht weiter. Die entsprechenden Bausteine reagieren auf diese Anforderung und stoßen die Übertragung an. **insert graphic of DB here**.
Der Schnittstellen-DB für die eingehende Kommunikation besitzt analog zum Ausgehenden zwei Datenbereiche für interne und externe Kommunikation, wobei hier beide als Byte-Arrays realisiert sind. Im Unterschied zum vorherigen DB enthält dieser zwei getrennte Signalisierungsmerker für intern und extern erhaltene Daten.

5.1.2 Interne Kommunikation

Die Interne Kommunikation besteht aus Sicht der Wegfindung vor allem aus dem Empfangen von Positionsdaten der zugeordneten Fahrzeugsteuerung und Senden der berechneten Routen an Selbige. Die Fahrzeugsteuerung erkennt durch den vorgelagerten RFID-Sensor den voraus liegenden Knoten und teilt diesen der Wegfindung mit. Da sich beide Schichten auf der gleichen Steuerung befinden

besteht hier die interne Kommunikation der Einfachheit halber nur aus dem Kopieren der Positionsdaten aus dem ausgehenden Schnittstellen-DB der Fahrzeugsteuerung in den entsprechenden Speicherplatz im eingehenden DB der Wegfindungsschicht. Um die Wegfindungsschicht über die neuen Daten in Kenntnis zu setzen wird das interne Signalisierungsbit im eingehenden Schnittstellen-DB der Wegfindung einen Zyklus lang gesetzt.
Die Positionsdaten sind haben den folgenden Aufbau:

Arrayindex	**Bedeutung**
1	ID des sendenden Fahrzeugs.
2	ID des erkannten aktuellen (RFID-)Knotens.
3	ID des nächsten Knotens entlang der aktuellen Route.

Die ID des sendenden Fahrzeugs wird für die interne Kommunikation nur dann benötigt, wenn sich die Wegfindung nicht auf der selben Steuerung wie die Fahrzeugsteuerung befindet. Durch die ID des aktuellen Knotens wird die bevorstehende Kreuzung identifiziert. Anhand des Knotens an Index 3, zu dem an der bevorstehenden Kreuzung nach der derzeitigen Route abgebogen wird, kann die Teilstrecke[1] bestimmt werden die das FTF voraussichtlich als nächstes befahren wird. Diese Angabe ist vor allem für die Kollisionsvermeidung in Abschnitt 5.2 von Bedeutung. Stimmen die beiden KnotenIDs der Positionsdaten überein, so bedeutet dies, dass der Zielknoten einer Teilroute erreicht wurde. Gemäß Abschnitt 4.5.3 wird hiermit die Simulation der Bearbeitungsdauer gestartet und die Berechnung der nächsten Teilroute angestoßen.
Um eine berechnete Teilroute an die Fahrzeugsteuerung zu übersenden, wird diese zunächst im ausgehenden Schnittstellen-DB abgelegt. Durch Setzen des Signalmerkers wird ein Kopiervorgang, analog zu dem beim Erhalten der Positionsdaten, gestartet und nach Abschluss der Übertragung das Signalisierungsbit des eingehenden Schnittstellen-DBs der Fahrzeugsteuerung gesetzt. Dies signalisiert, dass dem Fahrzeug eine neue Teilroute zur Verfügung gestellt wurde.

5.1.3 Externe Kommunikation

Mittels der externen Kommunikation, werden die eigenen Positionsdaten den anderen Fahrzeugen in der gleichen Anlage mitgeteilt. Da die Übermittlung von Positionsdaten die Aufgabe der Fahrzeugsteuerung ist, werden von der Wegfindungsschicht nur Daten empfangen und keine versandt. Da die Fahrzeuge mobil sein müssen, wurde zur Datenübertragung an andere Fahrzeuge eine drahtlose Verbindung ausgewählt. Hierfür verfügt jedes FTF über sein eigenes WLAN-Client-Modul. Als Übertragungsprotokoll wurde das User Datagramm Protocol (UDP) gewählt, da im Rahmen dieses Protokolls ungerichtete Verbindungen

[1] Verbindungskante zwischen aktuellem und nächstem Knoten.

aufgebaut werden können. Dies hat den Vorteil, das mittels einer ungerichteten Verbindung die Broadcast-Adresse des Netzwerksegments als Verbindungspartner projektiert werden kann.
Diese Broadcast-Adresse ist eine reservierte Spezialadresse innerhalb eines Subnetzes, welche die Eigenschaft hat, das sie von allen Teilnehmern im gleichen Netzwerkabschnitt empfangen und ausgewertet wird. Diese Eigenschaft wird im vorliegenden Fall ausgenutzt, um Informationen an alle Netzwerkteilnehmer weiter zu leiten, ohne die spezifischen Netzwerkadressen der Teilnehmer kennen zu müssen. Dies hat den Vorteil, dass es gleich ist, welche anderen FTF sich zu einem bestimmten Zeitpunkt in der Anlage befinden. Solange die Fahrzeuge mit dem gleichen Netzsegment verbunden sind, können sie Daten von anderen Fahrzeugen empfangen und eigene Positionsdaten senden. Das gleiche gilt auch für Überwachungs- und Visualisierungssysteme, die einfach in das gleiche Subnetz eingehängt werden können um den Netzverkehr abzuhören, ohne den Betrieb der Anlage zu behindern.
insert network graphic here
Über diese Verbindung übermittelt jedes Fahrzeug die gleichen Informationen wie bei der internen Kommunikation an das Netz. Die somit erhaltenen Positionsdaten der anderen Fahrzeuge werden in der in Kapitel 4.5.1 eingeführten Liste der erfassten Fahrzeugdaten gesichert. Diese enthält somit Informationen über die zuletzt empfangene Position eines anderen Fahrzeugs. Sollte ein Fahrzeug den Sendevorgang eines anderen FTF nicht mitbekommen, so wird spätestens beim nächsten Sendevorgang die Position aktualisiert.

5.1.4 Vorteile der Modularität

Der modulare Aufbau des Anwenderprogramms hat nicht nur den Vorteil, dass die Wegfindungsschicht auf eine zentrale Steuereinheit ausgelagert werden kann. Vor allem bei der Kommunikation können durch die Modularisierung verschiedene Kommunikationsarten einfach und schnell gegeneinander ausgetauscht werden. Im Verlauf der Entwicklung des Anlagenmodells wurden verschiedene Übertragungsarten auf ihre Vor- und Nachteile hin untersucht. Folgende Kommunikationsmöglichkeiten wurden getestet:

- **UDP:**
  - **Vorteil**: Gleichzeitige Adressierung aller Fahrzeuge ohne Wissen derer Adressen möglich.
  - **Nachteil**: Ungerichtete Verbindung gibt keine Informationen über den Erhalt der Daten zurück.
- **TCP:**

  - **Vorteil**: Durch Verbindungsüberwachung werden Daten bei Verlust automatisch neu versendet. Geeignet für gesicherte Übertragung von Routen.
  - **Nachteil**: Eigene dedizierte Verbindung nötig zu jedem anderen Netzteilnehmer. Auf den kleinen S7-1200er Steuerungen aufgrund der auf acht beschränkten Anzahl gleichzeitige programmierter Verbindungen[4] nur begrenzt einsetzbar.

- **UDP-Broadcast Master-Slave:**

  - **Vorteil**: Erreichbare Teilnehmer werden detektiert erfolgreiche Übertragung wird bestätigt.
  - **Nachteil**: Nur die Masterstation kann senden. Wechseln des Masters dauert mit zirka 200ms verhältnismäßig lange und ist deshalb nur für wenige Teilnehmer realisierbar.

- **Internes Kopieren:**

  - **Vorteil**: Schnelles, fehlersicheres Verfahren.
  - **Nachteil**: Nur für Kommunikation auf gleicher Steuerung einsetzbar.

Durch die Unabhängigkeit der Schichten kann die Funktionalität der Modellanlage innerhalb gewisser Grenzen an die gegebenen Anforderungen angepasst werden, ohne die Funktionalität anderer Schichten zu beeinträchtigen.

5.2 Algorithmische Kollisionsvermeidung

Wenn sich mehrere FTF zur gleichen Zeit in der Anlage befinden, muss für eine funktionierende Anlage verhindert werden, das sich die Wege dieser Fahrzeuge kreuzen. Es muss somit verhindert werden, dass zwei Fahrzeuge zeitgleich den selben Streckenabschnitt befahren. Die Fahrzeuge verfügen an der Vorderseite über einen Reflex-Lichtschranken-Sensor zur Kollisionserkennung. Dieser hält das FTF bei Erkennung eines Hindernisses an, indem die Motoren abgeschaltet werden. Verschwindet das Hindernis, setzt das Fahrzeug selbstständig die aktuelle Route fort. Da dieser Sensor bei der Erkennung eines voran fahrenden Fahrzeugs ein mögliches Auffahren verhindert, ist es somit kein Problem, wenn beide Fahrzeuge auf dem gleichen Streckenabschnitt hintereinander in die gleiche Richtung fahren. Da ein FTF jedoch nur an Knotenpunkten die Richtung ändern kann, muss unbedingt verhindert werden, dass sich zwei Fahrzeuge auf dem gleichen Streckenabschnitt entgegen kommen. Ansonsten würden sich je nach Einstellung der Reichweite des Kollisionssensors entweder die Fahrzeuge gegenseitig erkennen und permanent stehen bleiben, oder die niedrigen RFID-Ausleger stoßen zusammen, da sie zu weit unten angebracht sind, um von dem

Kollisionssensor erkannt zu werden.
insert sideview of vehicle with sensors here

5.2.1 Generelle Vorgehensweise bei Kooperativen Wegfindungsalgorithmen

Um die Vorgehensweise der algorithmischen Kollisionsvermeidung klar darstellen zu können, ist es von Vorteil, zunächst die Grundzüge der kooperativen Wegfindung zu erläutern. Vor allem in Computersimulationen und Computerspielen werden häufig Algorithmen verwendet, welche die Bewegungen mehrerer Einheiten koordinieren und aufeinander abstimmen.
Um sicher zu stellen, dass Einheiten einen optimalen Weg nehmen, ohne sich gegenseitig zu behindern, ist es nötig, die Wege aller beteiligten Akteure zu kennen und diese in den Wegfindungsalgorithmus mit einfließen zu lassen. Die Berechnung des Weges stützt sich hierbei auf eine 3-dimensionale Matrix, bei der die ersten beiden Dimensionen die Topologie und die dritte Dimension die Zeit darstellen. Da sich Einheiten getaktet fortbewegen kann somit jeder Einheit innerhalb dieser Matrix zu jedem Zeitpunkt ein bestimmter Index zugeordnet werden[15].**insert graphic space-time matrix here** Diese Vorgehensweise kann verglichen werden mit der Reservierung eines bestimmten Knotenpunktes zu einem gegebenen Zeitpunkt, zu dem dieser Knoten für andere Einheiten blockiert ist[16].
In der hier realisierten Modellanlage ist diese Vorgehensweise jedoch nur begrenzt anwendbar. Da die Wegberechnung auf die einzelnen Fahrzeuge verteilt ist und nicht zentral gesammelt durchgeführt wird, müsste jedes Fahrzeug immer seine komplette Teilroute allen Anderen mitteilen. Dies ist theoretisch möglich, würde aber Kommunikationslast deutlich erhöhen somit die Skalierbarkeit der Anlage verschlechtern.
Ein weiterer Unterschied ist, dass im vorliegenden Anwendungsfall nicht die Graphenknoten reserviert werden müssten, sondern die Verbindungskanten, da diese die blockierten Streckenabschnitte darstellen. Da die Anlage um einiges mehr Kanten besitzt als Knoten, ist es aufgrund der Speicherbeschränkungen der leistungsschwächeren Steuerungen nicht möglich, Matrizen mit einer hohen Zeitgranularität zu verwenden.

5.2.2 Problem des Zeitlichen Indeterminismus

Das größte Hindernis bei der Implementierung eines puren kooperativen Wegfindungsalgorithmus ist jedoch das Taktungsproblem. Anders als bei den in Abschnitt 5.2.1 vorgestellten Anwendungsgebieten bewegen sich die einzelnen Fahrzeuge nicht in konstanter Zeit von einem Streckenabschnitt zum Nächs-

ten. Systembedingt sind haben nicht alle Teilstrecken die gleiche Länge. Zudem schwankt die Fahrgeschwindigkeit einer einzelnen FTF je nach Ladezustand der Akkus sehr stark. Dies macht Vorhersagen über die Position eines Fahrzeugs sehr ungenau, je weiter man in der Zeit vorausschaut.
Eine Möglichkeit, die Vorhersagen zu verbessern wäre, anhand von Zeitmessungen Werte für die Belegdauer von Teilstrecken zu bestimmen und diese dann als Basis für die Vorhersagen zu benutzen. Dies birgt jedoch das Problem, dass das Verhältnis von Fahrtzeit auf einer Strecke zu der Zeit die benötigt wird um an einem Knoten ab zu biegen nicht vernachlässigbar ist. Somit müssten für jede einzelne Verbindungsstrecke mehrere Messungen gemacht werden, die alle Abbiegemöglichkeiten beinhalten. Bei der Vorhersage muss in diesem Fall aus den verschiedenen Messungen diejenige ausgewählt werden, die für diese Kombination aus Teilstrecken die Richtige ist. Hierbei würde aber immer noch der Ladezustand der Akkus nicht berücksichtigt werden, der wie bereits erwähnt zu starken Geschwindigkeitsunterschieden führen kann.
Bei genauerer Betrachtung lässt sich aussagen, dass als brauchbare Vorhersage für die Position nur die Beschränkung auf die unmittelbare Zukunft sinnvoll ist. Dies bedeutet das beim erreichen eines Entscheidungsknotens nur mit Sicherheit gesagt werden kann, auf welcher der vier möglichen Verbindungsstrecken sich das Fahrzeug als nächstes befinden wird. Bei einem dynamischen System kann es vorkommen, das die geplante Route am nächsten Knoten bereits wieder verworfen wird und somit etwaige Vorhersagen über die erste Verbindungskante hinaus nicht mehr gültig sind.
Aus diesem Grund wurde entschieden, die dynamische Wegfindung auf der Basis einer einstufigen Positionsvorhersage zu implementieren.

5.3 Verhinderung von Deadlock-Situationen

Um eine funktionsfähige Anlage zu implementieren, muss verhindert werden das sich Fahrzeuge auf der gleichen Teilstrecke entgegenkommen. Um dies zu erreichen muss der Wegberechnungsvorgang so modifiziert werden, das die in Abschnitt 5.2.2 eingeführte einstufige Positionsvorhersage bei der Wegfindung berücksichtigt werden. Da die Fahrzeuge wie in Abschnitt 5.1.3 beschrieben Zugriff auf die Streckenbelegungen der anderen Fahrzeuge ist dies ohne weitere Informationen bereits möglich. Bei der Aktualisierung der Positionsdaten, welche von anderen Fahrzeugen erhalten werden, wird nun zusätzlich die Position diese Fahrzeugs in einer quadratischen Matrix von der Größe der Anlagentopologie festgehalten. Diese Matrix stellt die in Abschnitt 5.2.1 beschriebene dritte Dimension dar, jedoch mit der Begrenzung auf die Schritte t_0 und t_{next}.

5.3.1 Anpassung der Wegfindung

Durch die Einführung dieser Belegungsmatrix ist es nun möglich bei der Wegberechnung zu prüfen ob eine Teilstrecke der aktuellen Route durch ein anderes Fahrzeug besetzt ist. Dies ist jedoch nur sinnvoll für die jeweils nächste Verbindungsstrecke der aktuellen Teilroute, da weiter entfernte Strecken wahrscheinlich bereits wieder frei sind, bis das FTF sie erreicht. Somit muss der Wegfindungsalgorithmus überprüfen, ob der die Verbindungsstrecke zum ersten Knoten entlang des kürzesten Pfads besetzt ist, und falls dies der Fall ist einen Pfad ohne diese Verbindungskante ermitteln. Dies wird erreicht, indem dem Wegfindungsbaustein als zusätzliches Argument die ID desjenigen Knotens übergeben wird, der nicht durch einen Teilpfad der Länge eins erreicht werden kann. Bei der Wegberechnung wird dies durch Sperrung der Teilstrecke implementiert.
Es kommt nun zusätzlich hinzu, dass die Wegberechnung an jedem Knoten der aktuellen Route überprüfen muss, ob die jeweils nächste geplante Teilstrecke durch ein entgegenkommendes Fahrzeug belegt ist. Ist dies nicht der Fall so kann die vorher berechnete Route weiterhin verwendet werden. Ist das nächste Teilstück der Anlage jedoch belegt, muss der Weg unter dieser Voraussetzung neu berechnet werden. Dies entspricht der in Kapitel 3.2.2 beschriebenen Mischform aus Planungs- und Ausführungsphase des A*-Algorithmus, bei dem an jedem Knoten eine Route unter Berücksichtigung der Umgebung geplant wird.
insert graphic recalculate route here

5.3.2 Kommunikation verifizierter Teilstrecken

Die Möglichkeit, das sich die als nächstes geplante Teilstrecke bei dem Erreichen eines Knotens ändern kann, hat zur Folge, dass auch die externe Kommunikation mit anderen Fahrzeugen modifiziert werden muss. Es kann nicht länger bei Erkennung einer Kreuzung durch den RFID-Sensor parallel die Positionsdaten mit der geplanten Teilstrecke an die Wegfindungsschicht und anderen Fahrzeuge weitergeleitet werden. Dies würde dazu führen das nach einer notwendigen Neuberechnung durch die Wegfindung das Fahrzeug nicht mehr die vorher publik gemachte Teilstrecke befährt, sondern eine andere Verbindungsstrecke auf der neuen Route belegt. Dadurch würden entgegenkommende Fahrzeuge wissen, das diese Strecke eigentlich bereits belegt ist.
Eine Möglichkeit ist, bei einer Änderung der geplanten Route die aktualisierten Positionsdaten erneut zu senden. Dies erhöht aber die Kommunikationslast und ist deshalb nur begrenzt sinnvoll. Als Lösung wurde interne von der externen Kommunikation getrennt und zeitlich gestaffelt. Das aktualisierte Kommunikationsmodell sieht vor, das bei der Erkennung des Knotens zunächst nur die Wegfindung die geplante nächste Teilstrecke erhält. Diese verififiziert die Route und sendet in jedem Fall eine aktualisierte Route an die Fahrzeugsteuerung zurück.

Diese besteht entweder aus dem Teil der noch gültigen Route, der vom aktuellen Knoten zum Zielknoten führt, oder aus der neu berechneten Ausweichroute. **insert graphics for both options here.**
Hat die Fahrzeugsteuerung eine Route von der Wegfindung erhalten, so teilt sie die aktualisierte nächste Teilstrecke den anderen Fahrzeugen mit. Erhält die Fahrzeugsteuerung nicht sofort eine Route[2], so bewegt sie sich langsam weiter auf die Kreuzung zu, bis der Punkt erreicht ist, an dem die letzte Möglichkeit besteht die Route noch zu ändern. Dies ist der Zeitpunkt an dem sich das Fahrzeug genau auf der Kreuzung befindet. Hat das Fahrzeug bis zu diesem Zeitpunkt noch keine Rückmeldung von der Wegfindungsschicht erhalten, so bleibt es auf der Kreuzung stehen und wartet auf die Wegfindung. Die Wahrscheinlichkeit ist in diesem Fall hoch, dass die Verzögerung durch die Belegung des nächsten Streckenabschnitts durch ein anderes Fahrzeug verursacht wurde, die eine Neuberechnung der Route mit sich bringt. Erst wenn eine Route empfangen wurde, setzt sich das Fahrzeug wieder in Bewegung. **insert flow diagram with recalculation behavior here**
Um einen reibungslosen Ablauf der Anlage garantieren zu können, ist es somit wichtig, das die Neuberechnung der Route möglichst schnell ausgeführt wird. Aus diesem Grund wurde in Kapitel 4 ein so hoher Wert auf die Effizienz der Implementierung der Algorithmen gelegt, um einen Halt an der Kreuzung zu vermeiden.

[2] auch nicht die aktualisierte alte Route.

6

Fazit

6.1 Aktueller Stand der Anlage

6.2 Komplikationen bei der Implementierung

6.3 Ausblick über zukünftige Erweiterungen

7

Anhang

Literaturverzeichnis

[1] T. Bauerhansl, M. ten Hompel, and B. Vogel-Heuser, *Industrie 4.0 in Produktion, Automatisierung und Logistik*. Springer Vieweg, 2014.

[2] B. Pottler, "Industrie 4.0 - Modell zusammen mit CT." Version 3, Oct. 2015.

[3] A. Meier, "Entwicklung eines Fahrzeugs für ein Fahrerloses Transportsystem im Modellmaßstab," Bachelorarbeit, Hochschule für angewandte Wissenschaften München, Aug. 2016.

[4] Siemens AG, *Simatic S7-1200 Automatisierung Systemhandbuch*, June 2015.

[5] P. Chakrabarti, S. Ghose, A. Acharya, and S. de Sarkar, "Heuristic search in restricted memory," *Artificial Intelligence*, vol. 41, pp. 197–221, dec 1989.

[6] R. E. Korf, "Real-time heuristic search," *Artificial Intelligence*, vol. 42, pp. 189–211, mar 1990.

[7] A. Stentz, "Optimal and efficient path planning for partially-known environments," *Proceedings IEEE International Conference on Robotics and Automation*, Mai 1994.

[8] S. Koenig and M. Likhachev, "Fast replanning for navigation in unknown terrain," *IEEE Transactions on Robotics*, vol. 21, pp. 354–363, June 2005.

[9] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–270, 1959.

[10] M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," in *Proc. 25th Annual Symp. Foundations of Computer Science*, pp. 338–346, Oct. 1984.

[11] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, pp. 100–107, July 1968.

[12] A. Patel, "Amit's thoughts on pathfinding: Heuristics." `http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html`, 2016. Abgerufen am 05.08.2016.

[13] Siemens AG, *STEP 7 Professional V13 SP1 Systemhandbuch*, Dec. 2014.

[14] B. Cherkassky, A. V. Goldberg, and T. Radzik, "Shortest paths algorithms: Theory and experimental evaluation," *Mathematical Programming*, vol. 73, pp. 129–174, May 1993.

[15] D. Silver, "Cooperative pathfinding," *Association for the Advancement of Artificial Intelligence*, 2005.

[16] M. Erdmann and T. Lozano-Perez, "On multiple moving objects," in *Proc. IEEE Int. Conf. Robotics and Automation*, vol. 3, pp. 1419–1424, Apr. 1986.