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

RTA* Real-Time A*

SCL Structured Control Language

SPS Speicherprogrammierbare Steuerung

STEP7 STeuerungen Einfach Programmieren Version 7

TIA-Portal Totally Integrated Automation Portal

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 einmal erläutert werden, was Industrie 4.0 im Kontext der Automatisierungstechnik bedeutet. Der Begriff beschreibt die vierte industrielle Revolution und die damit einhergehende Verflechtung von informationstechnisch erhobenen Daten in den Produktionsablauf. Ein interessanter Aspekt ist hier beispielsweise 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 auch ein sogenanntes Cyber-physisches Produktionssystem (CPPS) von Bedeutung. Diese bestehen aus der Verbindung von einzelnen, dezentralen Objekten wie beispielsweise 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, das 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, zum Beispiel zu Vorführzwecken bei Kunden.

2.2 Aufteilung der Themenbereiche

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

noch immer ein recht allgemein gefasstes Themengebiet ist und der zur Realisierung notwendige Aufwand nur schwer abzuschätzen 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 von Herrn Andreas Meier bearbeitet und wird deshalb nur kurz umrissen.

2.2.1 Fahrerloses Transportsystem

Hauptaufgabe der Werkstückträger, die jeweils 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 die Navigation innerhalb der Anlagentopologie, da die einzelnen FTF nicht wie sonst üblich schienengebunden sind, sondern sich frei durch die Anlage bewegen können. Da es 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, 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:

- Persistente Blockaden und länger andauernde nicht permanente Änderungen der Standard-Anlagentopologie.

[1] Speicherprogrammierbare Steuerung (SPS)

- 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, es existiert pro Funktionalität nur eine Bearbeitungsstation. Mehrere FTF erhalten die gleiche Route und reihen sich hintereinander ein.

Stufe 2: Es wird 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 definiert, zur besseren Darstellung der einzelnen Aspekte der definierten Stufen. Die Anlage wird zudem in Kooperation mit der Siemens Entwicklungsabteilung Corporate Technologies (CT) entwickelt wird, 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 zuallererst die Topologie der besagten Anlage benötigt. Für die Funktionsweise der FTF wurde definiert, das 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. Dies bedeutet, das 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 Dijkstra-Algorithmus

Der erste von insgesamt zwei für die Wegfindung verwendeten Algorithmen ist der nach seinem Erfinder benannte Dijkstra-Algorithmus, der den kürzesten Pfad zwischen einem Startknoten und einem Endknoten[1] eines Graphen ermittelt. Er kann verwendet werden für beliebige positiv gewichtete, gerichtete oder ungerichtete Graphen. Da in 3.1 negative Kantengewichtungen ausgeschlossen wurden, lässt sich der Algorithmus ohne Abwandlung auf den Graphen der Anlagentopologie anwenden.

3.2.1 Grundprinzip

Das Grundprinzip des Dijkstra-Algorithmus ist Unterteilung aller Graphenknoten in drei Untergruppen[3]:

Gruppe A: Die Menge aller Knoten, zu dem bereits ein kürzester Pfad bekannt ist.

Gruppe B: Die Menge aller Knoten, die mit mindestens einem Knoten aus Gruppe A verbunden sind, jedoch selbst nicht zu A gehören.

Gruppe C: Die Menge der restlichen Knoten, die nicht in den Gruppen A oder B enthalten sind.

Zusätzlich zu den Knoten können auch die Kanten drei unterschiedlichen Teilgruppen zugeordnet werden[3]:

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

Gruppe 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 Kante für jeden Knoten aus Gruppe B.

Gruppe III: Die restlichen Kanten.

Der Algorithmus funktioniert nun wie folgt. Zu Beginn befinden sich alle Knoten und Kanten respektive in den Gruppen C oder III. Als erstes wird nun der Startknoten S zu A hinzugefügt, da ein kürzester Pfad vom Startknoten zu sich selbst mit dem Wert 0 bekannt ist. Ab nun werden folgende Schritte solange wiederholt, bis das Ziel erreicht wurde:

[1] oder allen anderen Knoten.

1. Man betrachte alle Kanten z die den soeben zu A hinzugefügten Knoten mit einem Knoten K aus B oder C verbinden.
   - Gehört K zur Gruppe B, so wird untersucht ob z zu einem kürzeren Pfad zu K führt als der bisherige kürzeste Pfad mit einer Kante aus II zu diesem Knoten. Ist dies der Fall, so ersetzt z die korrespondierende Kante aus II. Die Kante die durch z wird wieder zur Gruppe III hinzugefügt.
   - Gehört K zur Gruppe C, so wird K zu B und z zu II hinzugefügt.
2. Es existiert immer nur jeweils ein Weg vom Startknoten zu einem beliebigen Knoten von B unter Verwendung der Kanten aus I und II. Nun wird derjenige Knoten aus B zur Gruppe A hinzugefügt, dessen Weg der kürzeste ist. Analog wird die zugehörige Kante der Gruppe I zugeordnet.

3.2.2 Darstellung der Funktionsweise

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

3.2.3 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[4] 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.

3.3 A*-Algorithmus

Der zweite verwendete Algorithmus für die Wegfindung ist der sogenannte A*-Algorithmus. Dieser ist eine Weiterentwicklung des in 3.2.1 vorgestellten Dijkstra-Algorithmus. Der grundlegende Unterschied ist jedoch, das bei A* eine 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 als nächstes expandiert, der den kürzesten Abstand zu den bereits expandierten Knoten 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 wird auch als Heuristik bezeichnet. Mittels dieser Bewertung kann nun bereits vor Erreichen des Zielknotens eine Aussage über die Länge der Route gemacht werden.

3.3.1 Grundprinzip

Die Funktionsweise des A*-Algorithmus kann somit wie folgt als Erweiterung des Dijkstra-Algorithmus beschrieben werden[5]:

- *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 $f(n)$ eine Funktion zur Abschätzung der Pfadlänge vom Knoten s zum Knoten z, welcher den Knoten n enthält.*

1. Man markiere s als „offen“ und berechne $f(s)$.
2. Man wähle denjenigen offenen Knoten n aus, der den kleinsten Wert für $f(n)$ besitzt.
3. Falls $n = z$, wird n als „geschlossen“ markiert und beende den Algorithmus.
4. Andernfalls markiere man n als „geschlossen“ und füge alle von n 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.

Man sieht das somit bei dem A*-Algorithmus diejenigen Knoten bevorzugt betrachtet werden, die durch die Abschätzungsfunktion als „wahrscheinlicher zum Ziel führend“ bewertet wurden. Somit ist es nur sinnvoll diesen Bewertungsvorgang näher zu betrachten.

3.3.2 Abschätzungsfunktion f(n)

Die Abschätzungsfunktion, die in 3.3.1 definiert wurde, kann unterteilt werden in zwei Unterfunktionen[5]:

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

$g(n)$ beschreibt hierbei den Wert des kürzesten Pfads von s zu n und $h(n)$ beschreibt den Wert des Pfades von n zum Ziel z.

Da die genauen Werte für $g(n)$ und $f(n)$ im Verlauf der Berechnung aber möglicherweise noch nicht exakt bestimmbar sind [2], werden Schätzwerte für $g(n)$ und $h(n)$ verwendet. Sei somit $\hat{g}(n)$ der Wert des bisher gefundenen kürzesten Pfads von s zu n und sei $\hat{h}(n)$ der Wert der Heuristik des Pfads von n zum Ziel z so folgt als Abschätzung für $f(n)$

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

Man kann nun beweisen das für alle Knoten n, die als offen markiert sind, der kürzeste Pfad vom Startknoten aus bereits bekannt ist, und somit gilt:

$$\hat{g}(n) = g(n) \tag{3.3}$$

Da davon ausgegangen werden kann, das alle Pfade $g(k_i)$ für bereits geschlossene Knoten k bekannt und minimal sind und somit auch gilt[3]

$$\hat{g}(n) = g(k_i) + min(v_{k_i,n}) = g(n) \tag{3.4}$$

muss nur noch die Heuristik $\hat{h}(n)$ definiert werden. Es kann bewiesen werden, das der A*-Algorithmus konsistent ist, das heißt, dass jeder bereits geschlossene Knoten nicht erneut geöffnet werden muss, wenn gilt:

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

Für die Berechnung bedeutet dies, dass wir den Wert des Restpfades von n zu dem Zielknoten z durch die Heuristik zwar unterschätzen dürfen, falls wir aber für die Berechnung einen möglichst effizienten Algorithmus benötigen, wir die Heuristik auf keinen Fall überschätzen dürfen.

3.3.3 Vergleich verschiedener Heuristiken h(n)

Die in Kapitel 3.3.2 besprochene Abschätzungsfunktion $f(n)$ hat einen großen Einfluss auf das Verhalten des A*-Algorithmus. Zur Veranschaulichung sollen einige Möglichkeiten für die Heuristiken betrachtet werden[6]:

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

[2] exakt bestimmbar sind diese erst wirklich beim erreichen des Zielknotens z und sukzessivem Beenden des Algorithmus.

[3] analog zu 3.2.1 Schritt 2 mit $v_{k,n}$ als die Gewichtung der Kante zwischen den Knoten k und n.

aus der Menge von offenen Punkten ausgewählt wird, dessen Abstand zu einem der geschlossenen Knoten am kleinsten ist. Der kürzeste gefundene Pfad durch einen Zwischenknoten n hat zu jedem Zeitpunkt i immer $f(n) = g_i(n)$. Somit entspricht der A*-Algorithmus ohne zusätzliche Informationen genau dem Dijkstra-Algorithmus aus 3.2.1.

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 garantiert den kürzesten Pfad zwischen Start und Ziel. Je kleiner $\hat{h}(n)$ im Verhältnis zu $h(n)$ ist, desto mehr Punkte 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 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 garantiert werden, dass nicht kürzere Pfade zu bereits geschlossenen Knoten existieren. Somit müssten bereits geschlossene Knoten erneut geöffnet werden um den tatsächlich kürzesten Pfad zu finden. 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 Abschätzungsfunktion $f(n)$ nur noch abhängig von $\hat{h}(n)$ und die anderen Terme können 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 Anlage in kürzester Zeit der kürzeste Weg ermittelt werden.

3.3.4 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. Als Heuristik wurde hier die Entfernung in Luftlinie zwischen den jeweiligen Städten und der Zielstadt gewählt, da diese die Bedingung $\hat{h}(n) \leq h(n)$ erfüllt:
insert graphic here

3.3.5 Berechnungsaufwand

Der Berechnungsaufwand für den A*-Algorithmus lässt sich nur schwer festlegen, da wie in 3.3.3 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:

$$0 \leq \hat{h}(n) \leq h(n) \tag{3.6}$$

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 die Bedingung 3.6 erfüllt:

$$\mathcal{O}(|\mathcal{V}|^2) \geq \mathcal{O}(\hat{h}(n)) \geq \mathcal{O}(1) \tag{3.7}$$

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

3.3.6 Nachteile und Alternativen zum A*-Algorithmus

Ein großer Nachteil des A*-Algorithmus ist der hohe Speicherverbrauch. Dieser kommt daher, das für die laufende Berechnung immer sowohl die Liste der geschlossenen als auch die der offenen Knoten im Speicher gehalten werden muss. Dies kann vor allem bei Graphen mit einer großen Anzahl von Kanten im Vergleich zur Anzahl der Knoten ein Problem sein. Zudem ist der A*-Algorithmus unflexibel was dynamische Veränderungen der Anlage sowie die partielle Neuberechnung von Teilrouten betrifft. Es existieren Weiterentwicklungen des A*-Algorithmus die sich unter anderem mit diesen Problemen befassen:

Memory Bounded A* (MA*)[7]: 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*)[8]: 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*)[9][10]: 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.

Für die Implementierung der Wegfindungsaufgabe die in 2.2.2 definiert ist, wurde eine Kombination aus dem Grundalgorithmus A* und der Planungs- und Ausführungsphase von RTA* verwendet, bei der in regelmäßigen Abständen der bisher gefundene Teilweg verifiziert und bei Bedarf neu berechnet wird. Für Anlagen Höherer Komplexität wäre zudem aufgrund des begrenzt verfügbaren Speichers eine Implementation von MA* sinnvoll. Der D*-Algorithmus wurde wegen seiner höheren Komplexität und der Beschränkungen durch die Programmierumgebung nicht implementiert, ist aber eine effizientere Alternative zum A*-Algorithmus, die vor allem durch ihre Dynamik besticht.

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 ist. 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 Anlagen bestehend aus mehreren Antrieben und SPSen gemeinsam geplant und erstellt werden. Die aktuelle Version des TIA-Portals ist 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:

Organisationsbaustein (OB): Diese Bausteine bilden die Schnittstelle zwischen dem Betriebssystem 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 und der Baustein zur zyklischen Abarbeitung von Teilschichten des Programms.

Datenbaustein (DB): Datenbausteine dienen zur Speicherung von variablen Daten, die im gesamten Anwenderprogramm benötigt werden. 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 Daten internen Variablen zur Verfügung. Bei der Verwendung eines FBs wird bei dessen Initialisierung ein entsprechender Instanz-DB generiert, in dem 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 verwendet und INOUT-Variablen werden sowohl schreibend als auch lesend verwendet. Innerhalb eines Bausteins können sowohl temporäre als auch statische[2] Variablen zur Zwischenspeicherung von Variablenwerten während der Programmabarbeitung verwendet 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 Assemblerähnliche Sprache für generelle Programmieraufgaben, die 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, das für den kompletten Zyklus als Basis für die Werte der Eingänge benutzt wird. Während des Zyklus 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 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.

[2] persistent über Funktionsaufrufe hinaus

4.2 Beschreibung der Anlagentopologie

4.3 Implementierung des Dijkstra-Algorithmus

4.4 Implementierung des A*-Algorithmus

4.5 Einhaltung der Echtzeitbedingung

[11]

4.5.1 Ausführung bei Systemstart

4.5.2 Zyklische Ausführung

5

Besonderheiten der Dynamischen Wegfindung

5.1 Kommunikation

5.1.1 Interne Kommunikation

5.1.2 Externe Kommunikation

5.2 Algorithmische Kollisionsvermeidung

[12]
[13]rausnehmen falls originaltext nicht beschaffbar ist bis dahin

5.2.1 Problem des Zeitlichen Indeterminismus

[14]

5.2.2 Verhinderung von Deadlock-Situationen

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." Präsentation Siemens AG, Oct. 2015. Version 3.

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

[4] 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.

[5] 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.

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

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

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

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

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

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

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

[13] A. Zelinsky, "A mobile robot exploration algorithm," *IEEE Transactions on Robotics and Automation*, vol. 8, pp. 707–717, Dec. 1992.

[14] 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.