

Hochschule für Angewandte Wissenschaften Hamburg  
*Hamburg University of Applied Sciences*

## Verbundprojekt

Jan Gellermann, Inke Heynen und Kai Robin Möller  
Gewerk 2 : Bahnplanung

# Inhaltsverzeichnis

<b>Tabellenverzeichnis</b>	<b>4</b>
<b>Abbildungsverzeichnis</b>	<b>5</b>
<b>1 Einführung</b>	<b>7</b>
<b>2 Aufgabenstellung</b>	<b>8</b>
<b>3 Bahnplanung</b>	<b>9</b>
3.1 Lokalisierung . . . . .	9
3.2 Kartenerstellung . . . . .	10
3.3 Trajektoriengenerierung . . . . .	10
3.3.1 Globale Bahnplanung . . . . .	11
3.3.2 Lokale Bahnplanung . . . . .	13
3.4 Ausgewähltes Konzept . . . . .	16
3.4.1 Gesamtkonzept . . . . .	17
<b>4 Konzept</b>	<b>21</b>
4.1 Potentialfeld zur Bahngenerierung und Kollisionsvermeidung . . . . .	22
4.1.1 Roboter Umwelt . . . . .	22
4.2 Kollisionsvermeidung durch kritische Bereiche . . . . .	24
4.2.1 Selbstorganisation durch Wartebereiche . . . . .	24
4.2.2 Bekannte Hindernisse . . . . .	25
4.2.3 Unbekannte Hindernisse . . . . .	26
4.3 Schnittstellen . . . . .	27
4.3.1 Bahnregelung . . . . .	27
4.3.2 Fertigungsplanung . . . . .	28
4.3.3 Fehlererkennung . . . . .	29
<b>5 Simulation und Testplanung</b>	<b>30</b>
5.1 Simulation des Ausweichmanövers . . . . .	30
5.2 Simulation Wartebereiche . . . . .	33

<b>6 Implementierung und Zielsysteme</b>	<b>35</b>
6.1 Selbstorganisation durch Wartebereiche . . . . .	36
6.2 Bekannte Hindernisse . . . . .	38
6.3 Unbekannte Hindernisse . . . . .	38
6.4 Potentialfeld . . . . .	39
6.5 Funktionsblockbeschreibungen . . . . .	41
6.5.1 UDP Gewerk1 Kommunikation . . . . .	41
6.5.2 UDP Daten formatieren . . . . .	42
6.5.3 Fehler erkennen . . . . .	42
6.5.4 Auftrag steuern . . . . .	42
6.5.5 Position der Robotinos bestimmen . . . . .	43
6.5.6 Fifo bestimmen . . . . .	44
6.5.7 Geschwindigkeit bestimmen . . . . .	46
6.5.8 Fahrmodus steuern . . . . .	48
6.5.9 Geschwindigkeit begrenzen . . . . .	50
6.5.10 Geschwindigkeit transformieren . . . . .	50
6.5.11 Status bestimmen . . . . .	50
<b>7 Robotino 2.0</b>	<b>51</b>
<b>8 Validierung</b>	<b>52</b>
<b>9 Ausblick</b>	<b>54</b>
<b>10 Fazit</b>	<b>55</b>

# **Tabellenverzeichnis**

3.1 Bahnplanungsmethoden nach ihrer Dynamik, Sicherheit, Rechenzeit und Zuverlässigkeit bewertet . . . . .	16
4.1 Zuordnung der Farben aus Abbildung 4.2 . . . . .	23

# Abbildungsverzeichnis

3.1	Beispiel Sichtbarkeitsgraph, Graue Felder sind Hindernisse, blaue Linie ist optimaler Pfad . . . . .	11
3.2	Beispiel Occupancy Grid, Rot-Hindernisse, Weiß-befahrbarer Raum . . . . .	13
3.3	Beispiel Occupancy Grid mit D*, Rot-Hindernisse, Intensität des Graus im Hintergrunds repräsentiert die Entfernung zum Ziel; blauer Punkt entspricht Ziel .	14
3.4	Bu2-Algorithmus, Grauen Flächen: Hindernisse, Roter Punkt: Startpunkt, Grüner Punkt: Ziel, Verbindungslien: m-Linie . . . . .	15
3.5	Bu2-Algorithmus, Fahrweg des Roboters ist in orange dargestellt . . . . .	16
3.6	Bu2-Algorithmus, möglicher Deadlock, Fahrweg des Roboters ist in orange dargestellt . . . . .	17
3.7	Beispiel Potentialfeld . . . . .	18
3.8	Konfigurationsraum mit Wartepositionen, Fifo-Positionen, Stationen und Über gabepunkten . . . . .	18
3.9	Potentialfeld des Konfigurationsraumes eingezäunt in hohe Potentiale zur eindeutigen Begrenzung des Raumes, Stationen als ein hohes Potential dargestellt, Rampen und Fahrrinnen zum leiten der Robotinos, . . . . .	20
4.1	Bereichseinteilung . . . . .	21
4.2	Potentialfeldzonen . . . . .	23
4.3	Lageplan der FIFO Plätze . . . . .	24
4.4	Schnittstelle mit der Fertigungsplanung . . . . .	28
5.1	Simulation mit Ausweichfunktion . . . . .	31
5.2	Simulation ohne Ausweichfunktion . . . . .	32
5.3	Simulation der Wartebereiche . . . . .	34
6.1	Ablaufdiagramm Konzept . . . . .	35
6.2	Ablaufdiagramm der Selbstorganisation . . . . .	36
6.3	Ablaufplan des Subsystems Vektorberechnung . . . . .	41
6.4	Subsystem UDP Gewerk 1 . . . . .	42
6.5	Subsystem UDP Daten formatieren . . . . .	43
6.6	Stateflow Auftrag steuern . . . . .	44
6.7	Stateflow Wartebereich bestimmen . . . . .	45

6.8 Subsystem Vektorberechnung . . . . .	46
6.9 Fahrmodus steuern . . . . .	49

# 1 Einführung

Bei dem Stichwort „Autonome Systeme“ fällt der Gedanke schnell auf Industrie 4.0. Die vierte industrielle Revolution, wie sie von dem Bundesministerium für Wirtschaft und Energie genannt wird, beschreibt die selbstorganisierte Produktion durch intelligente und digitale Systeme. [?] Ein solches autonomes System soll als Produktionsstraße im Verbundprojekt der Hochschule für Angewandte Wissenschaft Hamburg mit Hilfe von Transportrobotern, sogenannten Robotinos, realisiert werden.

Zur Realisierung dieser Produktionsstraße werden bereits zu Beginn der Projektarbeit alle notwendigen Hardwarekomponenten zur Verfügung gestellt. Zu diesen Hardwarekomponenten gehören die Robotinos. Ein Robotino ist ein mobiles Robotersystem mit omnidirektionalem Antrieb, der es dem Roboter ermöglicht zu jeder Zeit in jede beliebige Richtung fahren zu können. Sie werden mittels ArUco-Marker und Deckenkameras lokalisiert. Bei den zu transportierenden Werkstücken handelt es sich um runde Bausteine. Diese Bausteine sind mit einem RFID-Transponder ausgestattet und können von den Lesegeräten an den Stationen gelesen werden. Insgesamt gibt es vier Stationen, die beidseitig angefahren werden können. Diese Stationen repräsentieren die Lager bzw. Maschinen, zu denen die Werkstücke, je nach Auftrag, transportiert werden müssen. Zwei zusätzliche Stationen sind als Ladestationen für die Robotinos ausgelegt.

Auf Grund der Komplexität dieses Projektes, wird das Gesamtprojekt in einzelne Aufgabenpakete unterteilt, welche in Gruppenarbeit von drei bis vier Personen zu bearbeiten sind. Insgesamt gibt es fünf Gewerke, darunter die Auftragskoordination, Bahnplanung und Regelung, deren Ziel es ist ein funktionsfähiges und zuverlässiges Gesamtsystem zu entwickeln. Zusätzlich wird sich zum Ziel gesetzt, eine neue Version des Robotinos, den Robotino 2.0, am Ablauf der Produktionsstraße zu beteiligen.

Um dieses Gesamtziel zu erreichen, sind gemeinsame Schnittstellen, stetige Kommunikation, sowie abgestimmtes Zeitmanagement von großer Bedeutung.

In der vorliegenden Dokumentation wird die Umsetzung der Bahnplanung eingehend erläutert. Zu den Aufgabenbereichen der Bahnplanung gehört die Vorgabe eines Weges für den Robotino von einem Start- zu einem Zielpunkt, sowie die Kollisionsvermeidung mit statischen und dynamischen Hindernissen.

## 2 Aufgabenstellung

Aufgabe der Bahnplanung ist es, den Robotino von einer beliebigen Startposition im bekannten Raum zu einem vorgegebenen Ziel fahren zu lassen. Dabei ist zu beachten, dass es zu keiner Zeit zu einer Kollision mit dynamischen oder statischen Hindernissen kommt.

Für die Aufnahme und Abgabe der Werkstücke, also das Werkstückhandling allgemein, muss sowohl zu der Auftragskoordination, wie auch zu der Regelung eine geeignete und zuverlässige Schnittstelle generiert werden. Dazu gehört auch das Anfahren an die Stationen und das Fehler-Handling.

Die Algorithmen zur Realisierung der Bahnplanung und Kollisionsvermeidung sind in Matlab/Simulink zu implementieren. Der interne Programmablauf ist als Simulink/Stateflow mittels Blockschaltbilder zu realisieren. Die zu erstellende Software wird auf jeden einzelnen Robotino geladen und läuft dort dezentral als xPC-Target.

# 3 Bahnplanung

Die Navigation, also das effiziente und kollisionsfreie Bewegen im Raum, gehört zu den Hauptaufgaben von autonomen mobilen Robotern. Um zum Beispiel Produktionsstraßen zukunftsähig und effizienter zu gestalten, müssen die Robotinos konkreten Aufgaben wie „Fahre zum Zielpunkt XY und nehme das Werkstück auf“ übernehmen und abarbeiten können. Dazu muss der Robotino zu jeder Zeit folgende Fragen beantworten können: „Wo befindet sich mich? Wo muss ich hin? Wie gelange ich dort hin?“[? ]

Anhand dieser Fragen lässt sich die Bahnplanung in drei Bereiche unterteilen:

- **Lokalisierung:** Genau Positionsbestimmung des Robotinos im bekannten Raum
- **Kartenerstellung:** Vom Roboter über Sensordaten erstellte oder durch Softwareimplementierung bekannte Karte des Raumes
- **Trajektoriengenerierung:** Berechnung von Trajektorien vom Startpunkt zum Ziel

## 3.1 Lokalisierung

Eine Bahnplanung kann nur dann erfolgen, wenn der Roboter seine eigene Position zu jeder Zeit lokalisieren kann. In dem Fall des hier ausgearbeiteten Projektes erfolgt die Lokalisierung über Deckenkameras, die mittels UDP-Kommunikation den Robotinos ihre eigene Position, aber auch die der anderen Robotinos im bekannten Raum senden. Intern wird über den Raum ein Koordinatensystem gelegt, so dass die genaue Position der Roboter in x- und y-Richtung ausgegeben werden kann. Zu diesem Zweck sind die Robotinos mit ArUco-Markern ausgestattet. Um ein zuverlässiges Gesamtsystem zu erschaffen und die Ausfallwahrscheinlichkeit zu minimieren, erfolgt die Lokalisierung zusätzlich über Positionsdaten des Gewerks 3 „Regelung“. Somit wird sichergestellt, dass bei ungenauen oder nicht vorhandenen Kameradaten die Robotinoposition zu jeder Zeit bekannt und ein unterbrechungsfreier Ablauf des Prozesses gegeben ist. [? ]

## 3.2 Kartenerstellung

Mit bekannten Karten und vom Roboter durch die Sensorik erstellte Karten helfen beim Wegfinden. Durch eine vorab erstellte Karte können Hindernisse und Sackgassen implementiert werden, die von dem Transportroboter gemieden werden müssen. Kennt der Robotino seine Umgebung durch eine Karte kann über Selbstlokalisierung ein optimaler Pfad generiert werden.[?] Im Fall des vorliegenden Projektes muss der Roboter nicht durch willkürliches Fahren im Raum vorab eine Karte von unbefahrbaren Punkten sammeln. Den Robotinos sind die festen Hindernisse, das heißt Lager, Lade- und Werkstationen, auf Grund der implementierten Programmierung von vornherein bekannt.

## 3.3 Trajektoriengenerierung

Durch die Bahnplanung können kollisionsfreie Trajektorien von einem Start- zu einem Zielpunkt generiert werden. Als Trajektorie wird in der Physik der Bewegungsverlauf des Roboters als Kurve im Raum bezeichnet, also die Zustandsänderung relativ zum Koordinatensystem über die Zeit. Es werden Solltrajektorien berechnet, die dem Gewerk 3 „Regelung“ über eine definierte Schnittstelle übergeben werden. Über diese Trajektorie wird der Robotino zum Ziel geführt.

Die Literatur bietet viele verschiedene Ansätze zur Berechnung des bestmöglichen Pfades. Die Auswahl eines solchen Ansatzes hängt von der projektspezifischen Aufgabenstellung und Definition von „bester Pfad“ ab. „Bester Weg“ wird im Allgemeinen mit kürzester Distanz assoziiert. Es kann aber auch andere Faktoren beinhalten, die angeben wie „teuer“ ein Weg ist. Ist der Untergrund zum Beispiel schlecht befahrbar, so könnte es schneller sein einen Umweg zu fahren. Auch könnten kinematische und dynamische Eigenschaften des Roboters in die Definition mit eingehen, so dass Pfade vermieden werden, die Kurven beinhalten, die der Roboter nicht fahren kann. Auch Kriterien wie Energieeffizienz, Schnelligkeit und größtmöglicher Abstand zu Hindernissen haben je nach Aufgabenstellung eine andere Gewichtung bezüglich des Bahnplanungsansatzes. Allgemein wird in der Bahnplanung zwischen globaler und lokaler Bahnplanung unterschieden.

Im Nachfolgenden werden unterschiedliche Bahnplanungsansätze betrachtet und bewertet.

### 3.3.1 Globale Bahnplanung

Bei der globalen Bahnplanung, in der englischsprachigen Literatur auch bekannt als „Map-Based Planing“ müssen dem Robotino der Raum und die sich darin befindlichen statischen Hindernisse und Sackgassen vollständig bekannt sein, um diese in jedem Fall zu vermeiden. Im Folgenden werden zwei Möglichkeiten der globalen Bahnplanung beschrieben:

- **Sichtbarkeitsgraph-Methode mit A\*-Algorithmus**
- **Occupancy Grid mit D\*-Algorithmus**

#### Sichtbarkeitsgraph Methode mit A\*

Der Sichtbarkeitsgraph (engl. Visible graphs) ist eine Methode um den kürzesten Pfad zwischen zwei Punkten zu finden. Diese Methode setzt voraus, dass vorab Start- und Zielpunkt, so wie Eckpunkte der statischen Hindernisse klar definiert und bekannt sind. Die Eckpunkte der Hindernisse werden dann durch eine Kante verbunden, wenn eine gerade Verbindungsline gezogen werden kann, ohne andere Hindernisse zu schneiden. Dadurch werden Eckpunkte zu Knotenpunkten. Die Hindernisse mit den Verbindungslien ist in Abbildung 3.1 dargestellt. [? ]

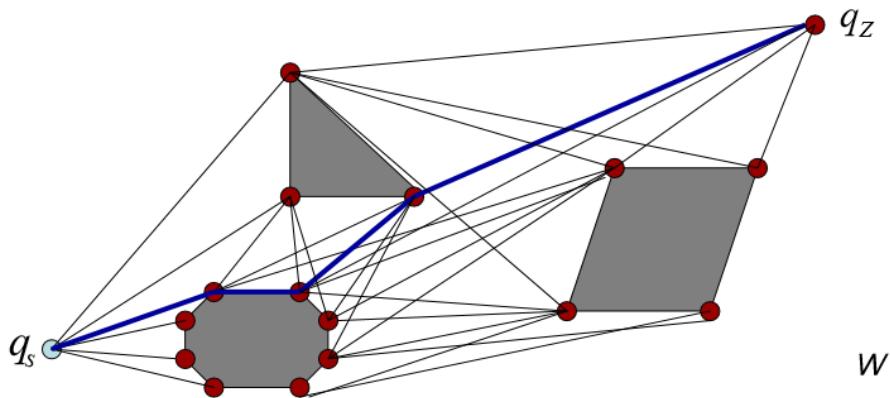


Abbildung 3.1: Beispiel Sichtbarkeitsgraph, Graue Felder sind Hindernisse, blaue Linie ist optimaler Pfad

[? ]

Durch Einsatz der A\*-Methode kann so der optimale Pfad bestimmt werden. Die Kanten zwischen zwei Knotenpunkten werden von dem Algorithmus je nach Anforderung, beispielsweise kürzester Weg, gewichtet. Je größer diese Gewichtung ist, desto weiter Weg befindet sich

der Knotenpunkt am anderen Ende der Kante. Die direkte Entfernung eines Knotenpunktes zum Zielpunkt wird geschätzt, so dass die Knoten ebenfalls eine Gewichtung bekommen.

Vom Zielpunkt aus werden nun alle Knoten betrachtet, die über eine Kante erreicht werden können. Der aus der Betrachtung hervorgehende Knotenpunkt, der die geringste direkte Entfernung zum Ziel aufweist, wird als nächstes untersucht. Die anderen werden gespeichert und im weiteren Verlauf mit Konten hinsichtlich ihrer Gewichtung verglichen. Punkte, die bereits betrachtet wurden, werden als „untersucht“ markiert. Dieses Verfahren wiederholt sich so lange bis der optimale Pfad gefunden wurde. Ein solcher Pfad ist in der obenstehenden Abbildung als dicke blaue Linie dargestellt.

Ein Vorteil dieser Methode ist, dass immer der optimale Pfad vom Start zum Ziel gefunden wird. Nachteile sind jedoch, dass der Pfad immer direkt an den Ecken und Kanten der Hindernisse entlang führt. Außerdem kann bei vielen Hindernissen, beispielsweise Hindernisse mit vielen Ecken und Kanten, ein sehr großer Graph entstehen, der viel Rechenzeit in Anspruch nimmt.

### Occupancy Grid mit D\*-Algorithmus

Wie der Name „Occupancy Grid“ vermuten lässt, wird der Raum in ein Gitternetz unterteilt. Jede entstehende Zelle kann mit einer „1“ für „belegt“ und „0“ für „frei“ versehen werden. Dadurch entsteht eine Umgebungskarte, in der Hindernisse und freier Raum zum Fahren klar definiert sind. In der Abbildung 3.2 ist ein Beispiel des Occupancy Grids dargestellt. Dabei sind anstatt Zahlen in den Zellen die freien Bereiche weiß und die besetzten Bereiche rot dargestellt. Der Übersichtlichkeit halber ist ein größeres Gitternetz dargestellt, als eigentlich unterteilt.[?]

Der D\*-Algorithmus verändert die Bewertung der einzelnen Zellen mit einem neuen Wert, der die „Kosten“ der Zelle repräsentiert. Zellen, die zu einem Hindernis gehören werden mit „ $\infty$ “ gewichtet. Je höher die Gewichtung ist, desto weiter weg befindet sich das Ziel. Mit diesem Algorithmus wird immer der optimale Pfad gefunden. Je nach Aufgabenstellung kann diese Gewichtung zum Beispiel Distanz oder Zeit bedeuten. In der nachstehenden Abbildung 3.3 ist der geplante Weg von dem Startpunkt zum Ziel mittels grün gepunkteter Linie dargestellt. Die Hindernisse sind rot markiert. Die Intensität des Grautons im Hintergrund gibt in diesem Fall die Entfernung zum Ziel an.

Ein Vorteil dieser Methode ist, dass der Weg schrittweise neu geplant werden kann. Sollte der Raum anders sein, als zuvor in dem Gitternetz eingetragen, eine Zelle beispielsweise teurer sein als geplant, so kann stufenweise ein besserer Pfad gefunden werden. Die Berechnung des neuen Weges erfordert nicht so viel Rechenleistung, wie eine komplett neue

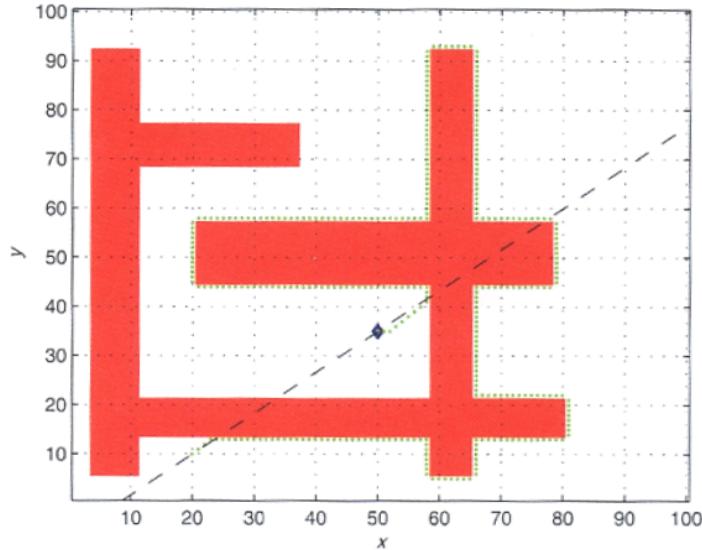


Abbildung 3.2: Beispiel Occupancy Grid, Rot-Hindernisse, Weiß-befahrbarer Raum  
[? ]

Wegplanung, da nur die Zellen in direkter Umgebung betrachtet werden. Die Anfangsrechnung ist hingegen sehr rechenintensiv. Zudem benötigt das Occupancy Grid bei hoher Auflösung viel Speicherplatz.

### 3.3.2 Lokale Bahnplanung

Die lokale Bahnplanung betrachtet nur das direkte Umfeld des Roboters. Lokale Bahnplanungsalgorithmen planen den Weg zum Ziel weniger vorausschauend. Daher benötigen sie in der Regel weniger Rechenzeit und sind somit schneller als globale Bahnplanungsmethoden. Mit den aktuellen Umgebungsdaten und Sensorinformationen wird ein lokales Navigationsziel angestrebt. Ziel ist hierbei die reaktive Kollisionsvermeidung. Im Folgenden werden zwei Möglichkeiten der lokalen Bahnplanung beschrieben:

- **Bug-Algorithmus**
- **Potentialfeld-Methode**

#### Bug-Algorithmus

Der „Bug-Algorithmus“ ist im Allgemeinen eine reaktive Navigationsmethode und basiert auf dem Prinzip, dass sich der Roboter so lange entlang eines Hindernisses bewegt, bis

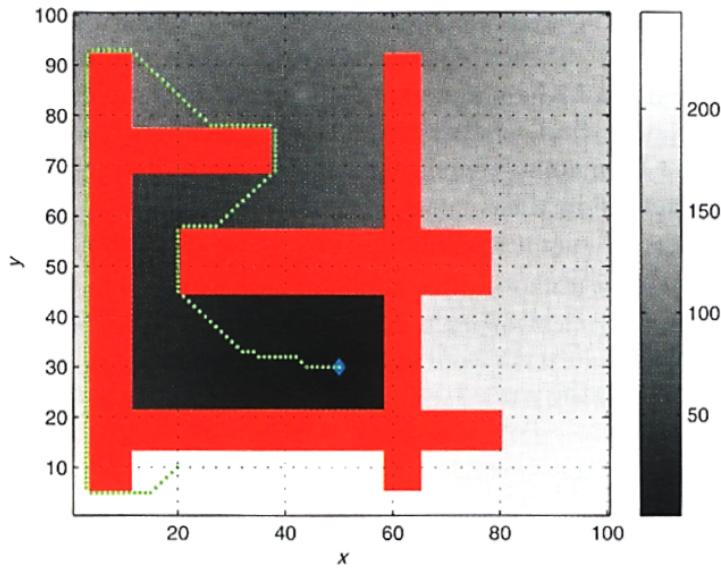


Abbildung 3.3: Beispiel Occupancy Grid mit D\*, Rot-Hindernisse, Intensität des Graus im Hintergrunds repräsentiert die Entfernung zum Ziel; blauer Punkt entspricht Ziel

[?]

er wieder freie Fahrt in Richtung Ziel hat. Es gibt verschiedene Algorithmen, die diese Methode verfolgen. Dazu gehört der „Bug1 -“, „Bug2-“, „Bug3-“ und „DistBug-Algorithmus“. Im Folgenden wird nur der Bug2-Algorithmus nähergehend erläutert. [?] [?]

Unter Verwendung des betrachteten Algorithmus kennt der Roboter zu jeder Zeit den direkten Weg zwischen seiner Startposition und dem Ziel. Hierbei werden mögliche Hindernisse zunächst nicht berücksichtigt. Abgesehen von einem Ziel im globalen Raum, kennt der Roboter nur seine direkte Umgebung. Der direkte Weg wird hier m-Linie genannt. Ein Beispiel ist in der untenstehenden Abbildung 3.4 dargestellt.

Der rote Punkt in der Abbildung ist die Startposition und der grüne Punkt das Ziel. Die Anweisungen des Algorithmus an den Roboter können in drei einfache Befehle unterteilt werden.

- *Fahre auf der m-Linie Richtung Ziel*
- *Ist ein Hindernis im Weg, dann fahre an dessen Kante entlang, bis die m-Linie wieder erreicht wird*
- *Ist die m-Linie wieder erreicht, verlasse das Hindernis und fahre weiter Richtung Ziel*

Wie ein möglicher Weg des Roboters unter dem Bug2-Algorithmus aussehen könnte, ist in Abbildung 3.5 zusehen. Die orangene Linie zeigt dabei den Fahrweg des Roboters

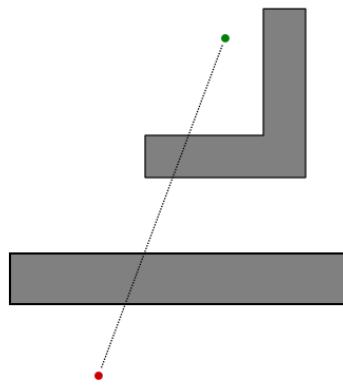


Abbildung 3.4: Bu2-Algorithmus, Grauen Flächen: Hindernisse, Roter Punkt: Startpunkt, Grüner Punkt: Ziel, Verbindungslien: m-Linie  
[? ]

Der Bug2-Algorithmus ist ein vergleichsweise einfacher Algorithmus. Das Entlangfahren an den Hindernissen kann nur in eine Richtung, links oder rechts, vorgegeben werden. Somit ist nicht garantiert, dass immer der optimale Pfad gefunden wird. Außerdem kann es zu einer Art „Deadlock“ kommen, wenn der Roboter an einer Kante des Hindernissen entlang fahren muss, die m-Linie jedoch nicht wieder findet. Ein solches Szenario ist in Abbildung 3.6 dargestellt.

### Potentialfeld-Methode

Bei der Potentialfeld-Methode ist der Roboter künstlichen Kräften von Zielen und Hindernissen ausgesetzt. Die hohen Potentiale stoßen den Roboter ab, niedrigere Potentiale ziehen ihn an. Jeder Punkt in dem Konfigurationsraum erhält ein Potential. So wird dem Start ein hohes und dem Ziel ein sehr niedriges Potential zugeordnet. Hindernisse bekommen sehr hohe Potentiale. Die Bewegungsrichtung des Roboters kann über die Gradientenberechnung des Potentialfeldes erfolgen, da der Roboter auf seinem Weg vom Start zum Ziel dem negativen Gradienten des globalen Potentialfeldes folgt. In die Berechnung des Gradienten gehen nur die Hindernisse mit ein, die sich in relativer Nähe zum Roboter befinden.[?] In der Abbildung 3.7 ist ein Beispiel eines Potentialfeldes dargestellt. Das niedrige Potential des Ziels ist in blau dargestellt.

Die Vorteile der Potentialfeld-Methode sind sowohl die einfache Implementierung als auch die geringe Rechenzeit. Durch die Echtzeit-Kollisionsvermeidung ist das System sehr dynamisch. Es besteht jedoch die Gefahr, dass lokale Minima entstehen, aus denen der Robotino nicht wieder raus kommt.

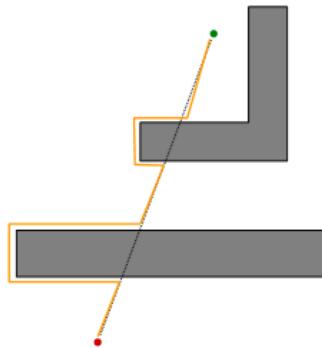


Abbildung 3.5: Bu2-Algorithmus, Fahrweg des Roboters ist in orange dargestellt  
[?]

### 3.4 Ausgewähltes Konzept

An der zu simulierenden Produktionsstraße sind bis zu fünf Robotinos gleichzeitig beteiligt. Um einen zuverlässigen Produktionsablauf sicherstellen zu können, werden hohe Anforderungen an die Bahnplanungsmethode gestellt. Die Methode muss dynamisch und zuverlässig sein und sollte geringen Rechenaufwand benötigen. Die soeben erläuterten Methoden sind in der Tabelle 3.1, reduziert auf die wichtigsten Eigenschaften, übersichtliche dargestellt.

Tabelle 3.1: Bahnplanungsmethoden nach ihrer Dynamik, Sicherheit, Rechenzeit und Zuverlässigkeit bewertet

Methode	Dynamik	Sicherheit	Rechenzeit	Zuverlässigkeit
Sichtbarkeitsgraph mit A*	gering	mittel	mittel/hoch	sehr hoch
Occupancy Grid mit D*	mittel	hoch	mittel/hoch	hoch
Bug-Methode	mittel	gering	gering	gering
Potentialfeld-Methode	sehr hoch	mittel	gering	hoch

Bei sehr hoher Dynamik und hoher Zuverlässigkeit benötigt die Potentialfeld-Methode nur wenig Rechenzeit. Damit ist sie von den betrachteten Methoden die beste. Das Potentialfeld hat zudem den Vorteil, dass ein komplett autonomes Gesamtsystem geschaffen werden kann, da es für die Trajektorienberechnung nicht notwendig ist das Ziel und die Fahrtroute der anderen Robotinos zu kennen.

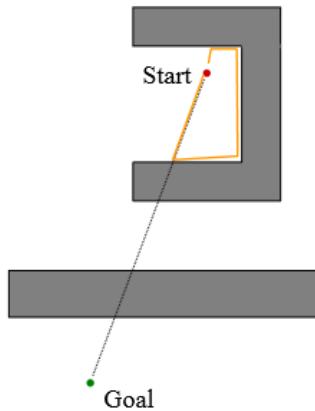


Abbildung 3.6: Bu2-Algorithmus, möglicher Deadlock, Fahrweg des Roboters ist in orange dargestellt

[?]

### 3.4.1 Gesamtkonzept

Im Folgenden wird das Gesamtkonzept mit der Bahnplanungsmethode Potentialfeld kurz erläutert. Genauere Erklärungen mit Auszügen aus dem Originalprogramm werden anschließend erläutert. Der Konfigurationsraum ist in Abbildung 3.8 mit globalem Koordinatensystem, eingezeichneten Stationen, Wartepositionen und Übergabepunkten dargestellt.

#### Wartepositionen

Jedem Robotino wird eine eigene Warteposition am Rand des Konfigurationsraumes zugeordnet. Diese Warteposition ist zu Beginn des Gesamtsystems die Startposition des jeweiligen Robotinos und immer dann Zielposition, wenn der Robotino keinen Auftrag empfängt. Sollten alle „Fifo-Plätz“ (vgl. Kapitel 3.4.1) einer Station belegt sein, so dass sich ein Robotino nicht mehr anstellen kann, muss er auf seiner Warteposition auf einen freien Platz warten.

#### Stationen und Übergabepunkte

Wie in Abbildung 3.8 zu sehen ist, befindet sich vor und hinter jeder Station Übergabepunkte. Hat ein Robotino die Aufgabe zu einer Station zu fahren, fährt er stattdessen nur vor die Station auf den Übergabepunkt. Ab da wird die Steuerung des Roboters an das Gewerk3 „Regelung“ übergeben. Sollten mehrere Robotinos die gleiche Station anfahren wollen, so darf der Robotino die Station zuerst befahren, der den Übergabepunkt zuerst erreicht hat.

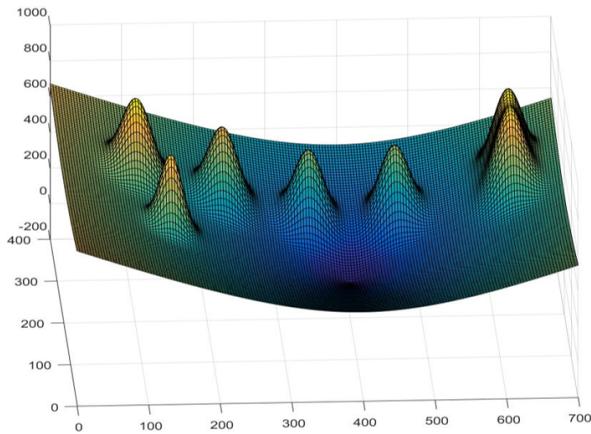


Abbildung 3.7: Beispiel Potentialfeld

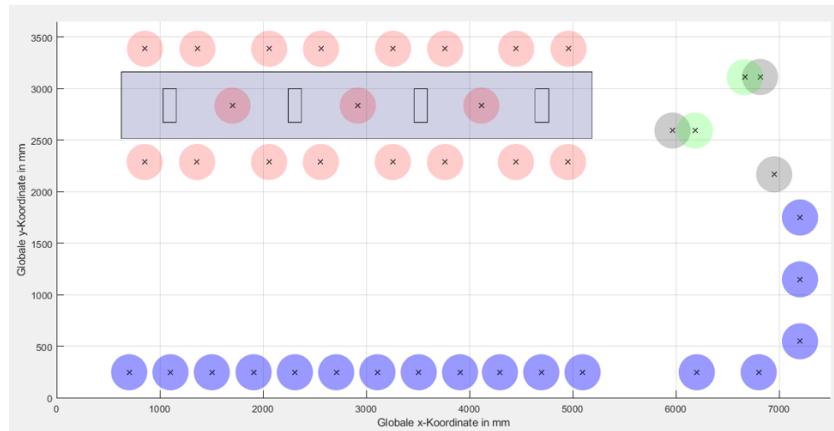


Abbildung 3.8: Konfigurationsraum mit Wartepositionen, Fifo-Positionen, Stationen und Übergabepunkten

Über die Übergabepunkte wird die Station auch wieder verlassen. Der vordere Übergabepunkt wird immer dann genutzt, wenn sich kein weitere Robotino im Fifo befindet. Über den hinteren Übergabepunkt wird die Station verlassen, wenn ein anderes Robotino bereits im Fifo (vgl. Kapitel 3.4.1) wartet. Das Potentialfeld ist in den Stationen, solange Gewerk3 „Regelung“ den Robotino steuert, nicht aktiv. Aktiviert beziehungsweise ausgeschaltet wird das Potentialfeld zum Zeitpunkt der Übergabe. Sollte ein Robotino, aufgrund eines notwendigen Ausweichmanövers, in eine Station gedrückt werden, so verlässt er sie unverzüglich nach hinten raus. In diesem Fall bleibt das Potentialfeld aktiv.

### Warteschlange „Fifo“

Ist eine Station, in die ein Robotino fahren soll, bereits belegt, so fährt er zu der zugehörigen Warteschlange, dem sogenannten „Fifos“. Die Fifos, die sich am Rand des Konfigurationsraumes befinden, sind Wartepositionen für die Stationen, um während des Wartens den befahrbaren Raum nicht zu blockieren. Der Robotino, der die Warteschlange zuerst betreten hat, darf diese, sobald die Station wieder frei ist, auch als erster wieder verlassen. Erst, wenn dieser die Station erreicht hat, rücken eventuell wartende Robotinos im gleichen Fifo auf.

### Ladestation

Die Ladestationen befinden sich an der rechten Seite des Konfigurationsraumes. Eine Ladestation ist für die Robotinos 1.0 und eine weitere für den Robotino 2.0. Auch hier sind Übergabepunkte definiert. Aus platztechnischen Gründen muss der Robotino 2.0 von hinten an die Ladestation fahren. Um dieses möglichst effizient zu realisieren, ist der Übergabepunkt für diese Ladestation vergleichsweise weit im eigentlichen Transportbereich. Ab diesem Punkt wird an Gewerk3 übergeben, welches den Robotino 2.0 mittels hoch genauer Regelung vor und in die Ladestation fahren lässt.

### Potentialfeld

Hindernisse im Potentialfeld werden mit hohen Potentialen versehen. Zu diesen Hindernissen gehören hier die Stationen und die Robotinos. In Abbildung 3.9 ist das gesamte Potentialfeld dargestellt, das auf jeden Robotino geladen wird.

Über eine Begrenzung durch hohes Potential wird der Raum, in dem sich die Robotinos bewegen dürfen, eindeutig definiert. Die Stationen sind als ein gesamter Block mit hohem Potential dargestellt, da der Robotino zu keiner Zeit zwischen oder an eine Station fahren soll, wenn nicht zuvor die Übergabe an Gewerk3 erfolgt ist. Zur Vermeidung von kritischen Situationen oder lokalen Minima zwischen Stationen und Wand sind Rampen und Fahrrinnen implementiert, die je nach Station den Roboter nach links oder rechts in eine Fahrrinne und von dort in den frei befahrbaren Raum führen.

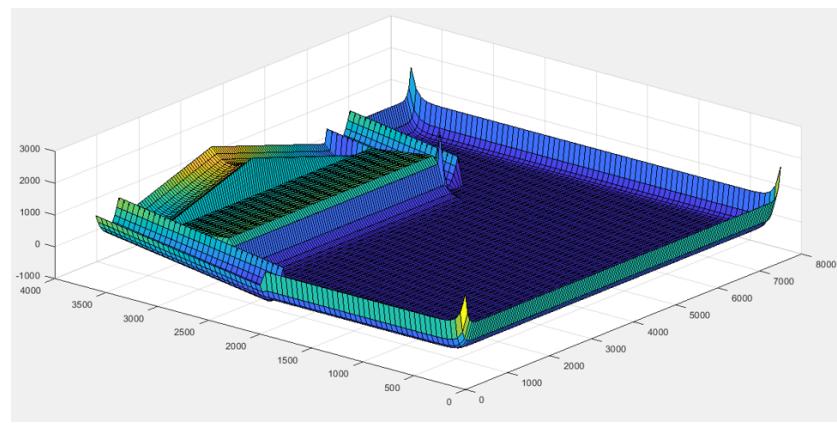


Abbildung 3.9: Potentialfeld des Konfigurationsraumes eingezäunt in hohe Potentiale zur eindeutigen Begrenzung des Raumes, Stationen als ein hohes Potential dargestellt, Rampen und Fahrrinnen zum leiten der Robotinos,

## 4 Konzept

In diesem Kapitel wird das in diesem Bericht genutzte Konzept beschrieben. Das Grundkonzept besteht darin, dass der zu befahrene Bereich in einen Fertigungsbereich und einen Transportbereich unterteilt wird. Die Grenzen der Bereiche sind in Abbildung 4.1 dargestellt. Im Transportbereich, in der Abbildung 4.1 nicht eingefärbt, wird die Regelung des Robotinos über die Potentialfeldmethode realisiert. Das dabei genutzte Potentialfeld wird unter Kapitel 4.1 näher erläutert. Im Fertigungsbereich wird die Regelung von der Bahnregelungsgruppe übernommen. Um zwischen den Bereichen zu wechseln, werden Übergabepunkte definiert, an denen der Bereichswechsel sicher ausgeführt werden kann. Diese Übergabepunkte sind in der Abbildung als rote Kreise dargestellt. Dazu wird eine Kommunikation zwischen den Einzelgruppen über eine Schnittstelle definiert. Diese Schnittstelle wird in Kapitel 4.3.1 definiert. Des Weiteren wird das Konzept zum Vermeiden von Kollisionen in Kapitel 4.2 näher beschrieben. Dabei wird auf verschiedene Hindernistypen eingegangen.

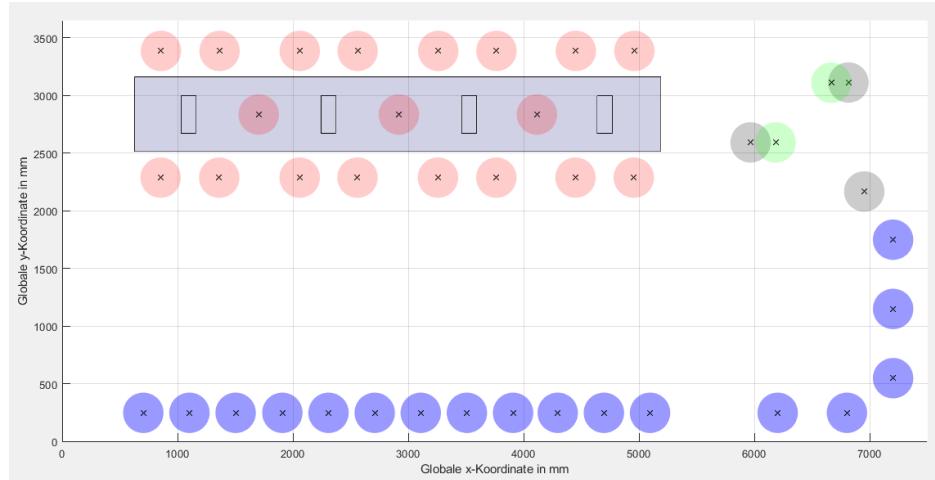


Abbildung 4.1: Bereichseinteilung

## 4.1 Potentialfeld zur Bahngenerierung und Kollisionsvermeidung

In diesem Abschnitt wird beschrieben, wie das Potentialfeld aufgebaut ist, das zur Generierung des Geschwindigkeitsvektors genutzt wird.

### 4.1.1 Roboter Umwelt

Um die statische Umwelt des Robotinos im Potentialfeld darzustellen, wird der zu befahrende Bereich in Zonen unterteilt. Die gewählten Zonen sind in Abbildung 4.2 farbig dargestellt. In Tabelle 4.1 ist die Zuordnung der Zonen zur Abbildung 4.2 beschrieben. Dabei wird eine Hauptfahrzone (Zone 0) definiert, in der mit einem virtuellen Zaun der zu befahrene Bereich abgegrenzt wird. Dieser Zaun ist notwendig, damit die Zone nicht unkontrolliert verlassen wird, dazu werden e-Funktionen genutzt, die in den Funktionen (??) bis (??) definiert sind. Aufgrund der Form der Zone müssen dabei drei Funktionen definiert werden, die je nach Position des Robotinos ausgeführt werden. Das Verlassen der Zone 0 erfolgt durch die Übergabe an den Fertigungsbereich, welches in Kapitel 4.3.1 näher erläutert wird.

Zone 1 bis 3 dienen zur Rückführung der Robotinos zur Zone 0, dazu wird ein Potentialfeld genutzt, dass uns ermöglicht den Robotino in eine gezielte Richtung zu lenken. Die dazu genutzten Funktionen sind unter (??) bis (??) definiert. Diese Potentialfelder nutzen in Fahrrichtung eine Geradengleichung mit definierter Steigung und senkrecht dazu eine Parabelform, um den Robotino auf Kurs zu halten. Wenn sich der Robotino in Zone 4 befindet, wird dieser über eine Geradengleichung nach hinten geführt. Die dabei genutzte Gleichung ist als (??) gekennzeichnet. Zusätzlich wird in Zone 4 für jede Station eine gaußsche radiale Basisfunktion, welche unter Kapitel ?? näher erläutert wird, genutzt, damit keine Kollision mit den Stationen auftreten. Diese Zone wird nur im Fehlerfall oder beim Start des Robotinos betreten, da in dieser Zone der Fertigungsbereich aktiv ist. Die Zone 5 dient dazu den Robotino gezielt in Richtung der Mitte der Zone 0 zu führen. Dazu wird, wie in Zone 1 bis 3, eine Parabel mit einer Geradengleichung genutzt. Die dabei genutzte Funktion ist als (??) definiert. Dabei ist zu beachten, dass vorgesehen ist, dass Zone 5 nur aktiv ist, wenn sie aus Zone 1 betreten wird. Dadurch wird die Anfahrmöglichkeit aus der Zone 0 zur Ladestation gewährleistet.

Je nach Zone gilt dabei ein eigenes Potentialfeld, welche kombiniert das Gesamtpotentialfeld ergebenen. Die einzelnen Zonen sind in Abbildung ?? dargestellt. Das gesamt Potentialfeld ist in Abbildung ?? dargestellt, wobei zur Übersichtlichkeit die Stations und Ladestationspotentiale nicht dargestellt werden. Da die Zone 5 nur durchlaufen wird, wenn sie aus Zone 1 betreten wird, wird in Abbildung ?? dargestellt, wie das Potentialfeld in diesem Fall aussieht.

Zone	Farbe	Beschreibung
0	ohne Farbe	Hauptfahrzone
1	Grün	Führung des Robotinos nach Zone 5
2	Blau	Führung des Robotinos nach Zone 3
3	Violet	Führung des Robotinos nach Zone 0
4	Rot	Führung des Robotinos nach Zone 1 und 2
5	Cyan	Führung des Robotinos nach Zone 0

Tabelle 4.1: Zuordnung der Farben aus Abbildung 4.2

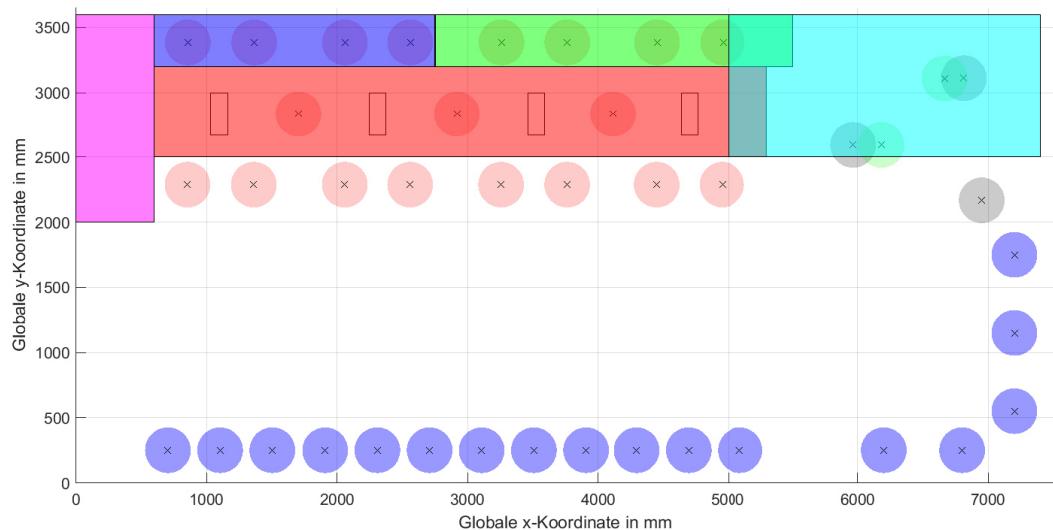


Abbildung 4.2: Potentialfeldzonen

## 4.2 Kollisionsvermeidung durch kritische Bereiche

Bei der Analyse des verwendeten Verfahrens zur Bahnplanung entstehen mehrere Bereiche, bei den die Potentialfeldmethode an ihre Grenzen stößt. Es treten lokale Minima auf, die nicht gleich mit dem Zielpunkt sind. Zwei Roboter können sich, bei gleichem Ziel, jeweils gegenseitig am Erreichen hindern. Hinter den Stationen und links von der ersten Station ist nicht genügend Platz zur perfekten Bahnplanung mittels Potentialen. Würde ein anderer Robotino in die gleiche Region fahren, ist die geplante Strecke nur noch schwer zu realisieren. Die Wahrscheinlichkeit, dass der Robotino gegen die Wand oder Station "gedrückt" wird steigt stark.

Aus diesem Grund werden hinter den Stationen Einbahnstraßen entworfen, in denen der Robotino geschützt wieder in den freien Bereich geführt wird. Hiermit wird jedoch noch nicht das Problem der Roboterbegegnung vor den Zielen verhindert.

### 4.2.1 Selbstorganisation durch Wartebereiche

Um die Entstehung von lokalen Minima zu verhindern, wird die Software der Robotinos um eine Selbstorganisation erweitert. Diese Erweiterung beinhaltet Wartepositionen, damit ein Roboter, der zur einer bereits besetzten Station fahren soll, sich außerhalb des Kollisionsbereichs anstellt.

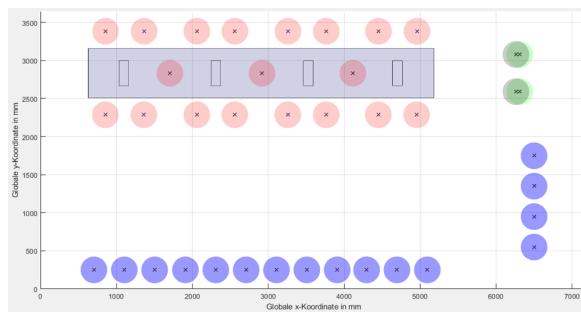


Abbildung 4.3: Lageplan der FIFO Plätze

In Abbildung 4.3 erkennt man am unteren Rand der Karte zwölf blaue Kreise, welche die Wartebereiche darstellen. Zu jeder Station gehören drei Wartepositionen. Durch sich verändernde Zählweisen, ist der zugehörige Warteplatz zu einer Station immer eindeutig zuzuweisen. Im Kapitel 6 wird dies weiter erläutert.

Mit Hilfe des UDP-Datenpaketes der Kameradaten ist es für einen Robotino möglich

die Positionen der anderen Robotinos zu erhalten. Ist nun die Zielstation bereits von einem anderen Robotino belegt, wird mit Hilfe der Selbstorganisationssoftware die erste mögliche Warteposition als Ziel übernommen. Ist diese auch bereits belegt wird die Warteposition um einen Platz erhöht.

Dem gesamten Anstellprozess liegt das „First In First Out-Prinzip“ (im weiteren nur noch FIFO genannt) zu Grunde. Dies bedeutet: Wer sich als erstes anstellt, darf auch als erstes wieder herausfahren.

Damit auch die Warteschlange beim Herausfahren eingehalten wird, wird die Software ein weiteres mal erweitert. Die Position der anderen Roboter wird in einem Merker gespeichert und erst zurückgesetzt, wenn der Roboter an der nächsten Warteposition, beziehungsweise an der Zielposition, angekommen ist. Besonders das Freischalten der ersten Warteposition ist kritisch. Wenn ein anderer Robotino die Zielstation verlassen hat, für diese Station aber mehr als ein Robotino wartet, würde beide Robotinos direkt zur Station fahren. Mit Hilfe des Merkers wird die erste Warteposition erst freigeschaltet, wenn der Robotino, der vorher in der ersten Warteposition war, an der Zielstation angekommen ist. Das bedeutet, dass auch während der längeren Strecke zwischen Wartepositionen und Stationen gesichert ist, dass nur ein Robotino gleichzeitig die Station anfährt. Ist der Roboter dort angekommen, wird die erste Warteposition freigeschaltet und der Robotino auf dem zweiten Platz darf auf den ersten Platz vorfahren.

Diese Warteschlange wird noch ein weiteres mal im Programmablauf abgefragt. Hat der Robotino seinen Auftrag abgearbeitet, also sein Werkstück abgeholt beziehungsweise abgelegt, wird die Belegung der Warteschlange zur zugehörigen Station abgefragt. Wartet dort bereits der nächste Roboter fährt der Robotino nach Hinter, also zur Wand hin, aus der Station heraus. Von dort gelangt er über die Einbahnstraßen wieder in das freie Feld.

Bei der Erweiterung des Projektes um einen fünften Roboter, wie in Kapitel 7 erklärt, wird die vierte Anstellposition nicht neben den anderen Wartepositionen generiert, sondern auf Grund von Platzmangel die Standbyposition einprogrammiert.

### 4.2.2 Bekannte Hindernisse

Um generelle Begegnungen anderen Robotinos in der normalen Fahrzone zu vermeiden, wird ein Ausweichmanöver entworfen. Befindet sich ein anderer Roboter im Bereich zwischen den gesteuerten Roboter und seinem Ziel und in einer Entfernung von 20 cm wird auf den Zielvektor eine 90°-Drehung addiert. Der Robotino fährt somit für eine kurze Zeit mit gleichbleibendem Abstand zum Ziel. Ist der behindernde Robotino nicht mehr im gefährdeten Bereich, wird wieder der normale Zielvektor eingestellt.

Für eine hohe Dynamik im Gesamtsystem unterscheidet die Software zwischen sich bewegenden und stehenden Robotinos. Anhand der Kameradaten werden die aktuellen Positionen mit deren vorherigen mittels Merker verglichen. Ist die Differenz größer der definierten Toleranz, bewegt sich der Robotino. Sich bewegende Robotinos stelle eine größere Kollisionsgefahr dar. Deshalb werden die Potentialfelder jener Robotinos vergrößert. Der gesteuerte Roboter wird somit in einem größeren Abstand vorbei fahren.

#### 4.2.3 Unbekannte Hindernisse

Das verbaute Kamerasystem ist nicht in der Lage andere Objekte, außer die mit Aruco-Markern versehenden Robotinos, zu erkennen. Um jedoch auch eine Kollision mit Menschen zu verhindern besitzt der Robotino neun Infrarot-Sensoren gleichmäßig um den untersten Rand des Roboters verteilt. Detektiert einer der Sensoren ein Objekt ermittelt die Software an welcher Stelle das Objekt erkannt wird und manipuliert den Zielvektor. Es wird ein abstößendes Potential an der Hindernisposition programmiert. Hat sich das unbekannte Hindernis entfernt, wird das Potential ab einem gewissen Abstand wieder abgeschaltet.

## 4.3 Schnittstellen

Im gesamten Projektablauf ist die Kommunikation mit anderen Gruppen ausschlaggebend für eine perfektes Gesamtkonzept. In der Bahnplanung wird vor allem mit der Gruppe der Bahnregelung und mit der Gruppe der Fertigungsplanung eng zusammengearbeitet.

### 4.3.1 Bahnregelung

In sehr engen Bereichen, in denen auch eine hohe Präzision erforderlich ist, stößt die Potentialfeldmethode an ihre Grenzen. Aus diesem Grund wird der gesamte Bereich direkt am Anfang des Projektes aufgeteilt. Das Potentialfeld wird in den freien Bereiche benutzt und an den Randbereichen des Feldes. Das Anfahren der einzelnen Fächer innerhalb der Stationen, sowie das Werkstückhandling, wird an die Bahnregelung übergeben.

In Abbildung 4.3 stellen die acht roten Kreise vor den Stationen die Übergabepunkte dar. Befindet sich der Robotino innerhalb eines solchen Kreises, inklusive einer gewissen Toleranz, wird ein Übergabebit gesetzt und die Bahnregelung übernimmt das präzise Anfahren zuerst des RFID Lesegerätes und anschließend das Ablegen in ein Stationsfach. (Beim Abholen ist der Prozess andersherum.) Somit gehören zur Schnittstelle nicht nur das Übergabebit. Die Gruppe der Bahnregelung benötigt darüber hinaus auch noch den genauen Auftrag der Fertigungsplanung. Hat die Bahnregelung den Auftrag abgearbeitet, wird der Übergabebit am Übergabepunkt getoggelt und der Robotino fährt wieder im Potentialfeld. Dieser Übergabepunkt ist variabel, wie bereits in dem Unterkapitel 4.2.1 erläutert.

Da die Bahnregelung sehr präzise die Position des Robotinos für eine hoch genaue Fahrt benötigt, wird von der Gruppe ein Kalman-Filter für die Robotino Position entworfen. Ohne Beobachter sendet die Kamera, bei nicht Erkennung eines Robotinos, die Positionsdaten [0,0] für diesen Robotino. Diese falsche Postion kann zu einigen Fehlern in den Berechnungen führen. Dieser Beobachter ermöglicht eine Fehlerfreie Positionsdarstellung des Roboters auch bei kurzzeitiger Verdeckung des ArUcomarkers. Die Beobachterdaten werden auch in der Bahnplanung durchgehende verwendet.

### 4.3.2 Fertigungsplanung

Die Fertigungsplanung ist für die gesamte Simulation der Fabrikumgebung zuständig. In der Gruppe werden verschiedene Aufträge generiert und intelligent auf die vier Robotinos verteilt. Mittels UDP-Kommunikation wird der jeweilige Auftrag an den jeweiligen Roboter gesendet. In Abbildung 4.4 ist der Informationsaustausch dargestellt. Die Informationen, die

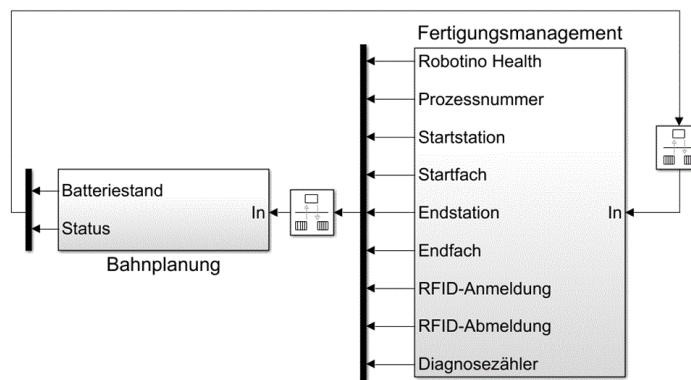


Abbildung 4.4: Schnittstelle mit der Fertigungsplanung

die Bahnplanung an die Fertigungsplanung zurücksendet, wird um einen weiteren Punkt erweitert. Die Ist-Position des Roboters wird ebenfalls gesendet. Diese Position wird von der Fertigungsplanung für die Fabriksimulation benötigt. Im Statusbyte wird übermittelt, was der Robotino zu dem Zeitpunkt gerade ausführt. Also ob er im Potentialfeld fährt oder eine Station anfährt, etwas ablegt oder etwas aufnimmt. Darüber hinaus werden im Status auch Fehlermeldungen weitergeleitet, dazu im Unterkapitel 4.3.3 mehr.

Von der Fertigungsplanung bekommt das Programm den genauen Auftrag, den der Robotino abarbeiten soll. Jeder neue Auftrag erhält auch eine neue Prozessnummer. Der Auftrag besteht aus der Startstation mit zugehörigem Fach, an dem ein Werkstück abgeholt werden und in welches Fach von welcher Station es gebracht werden soll. Damit jedes individuelle Werkstück auch entsprechend erkannt wird, sind an jeder Station RFID Lese-/Schreibköpfe angebracht. Wird ein Werkstück in die jeweilige Station gebracht, wird es dort angemeldet. Beim Aufnehmen wird es abgemeldet. Hierfür werden zwei Bits in der Schnittstelle verwendet. Die Information über die einzelnen Werkstücke werden von der Fertigungsplanung verarbeitet und in einer Cloud gespeichert.

### 4.3.3 Fehlererkennung

Eine weitere Übergabeinformation zwischen den drei Gruppen ist das Fehlerhandling. Es wird zwischen drei verschiedenen Fehlern des Robotinos unterschieden:

1. Kein Werkstück vorhanden
2. Stationsfach belegt
3. Werkstück verloren

Beim Anfahren eines Stationsfachs zur Aufnahme eines Werkstückes durch die Bahnregelung, wird zwischen den Greifern eine Lichtschranke abgefragt. Kann kein Werkstück aufgenommen werden, weil beispielsweise kein Werkstück erkannt wird, wird ein Fehler ausgelöst. Dieser Fehler wird anschließend an die Fertigungsplanung gesendet. Die Fertigungsplanung entscheidet dann, ob ein falscher Auftrag gesendet wird oder in der 'Fabrik' ein Fehler vorliegt, welcher per manuellen Eingriff behoben werden muss. Der zweite Fehler wird über einen Zähler der Anfahrversuche für ein Fach realisiert. Schafft es der Robotino innerhalb von drei Anfahrversuchen nicht das Werkstück abzulegen, so wird der zweite Fehler gesendet. Verliert der Roboter während der Fahrt das Werkstück, wird der dritte Fehler gesendet. Hier muss manuell das Werkstück aus dem Fahrbereich entfernt werden.

# 5 Simulation und Testplanung

Um das Potentialfeld auch ohne Robotinos zu testen, wird zu Beginn des Projektes mit einer vereinfachten Simulation das Potentialfeld auf Ihre Funktionsfähigkeit geprüft. Bei dieser Simulation wird der Robotino als einfache I-Strecke betrachtet. Dabei wird der Ausgang der Strecke auf dem Istpositionsinput gegeben und der berechnete Geschwindigkeitsvektor auf die Strecke gegeben. Des Weiteren werden auf die restlichen Inputs Konstanten gegeben und die Ausgangsdaten in einer Logdatei abgespeichert. Da nach wenigen Versuchen erkennbar ist, dass das Potentialfeld seine Funktion erfüllt, wird im Folgenden mit den Robotinos direkt getestet. Dazu wird zu Beginn der Fertigungsbereich nicht betrachtet und nur ein fest programmiertes Ziel angefahren. Um flexibel Ziele anfahren zu können und die Schnittstelle mit der Fertigungsplanung zu testen, wird ein UDP-Dummy in Simulink erstellt, mit dem Aufträge an den Robotino per UDP in Simulink gesendet werden können. Dadurch können mehrere Ziele hintereinander angefahren werden. Im nächsten Schritt wird die Schnittstelle mit der Bahnregelung getestet. Dazu werden beide Simulinkmodelle kombiniert und auf den Robotino geladen. Dadurch können Schnittstellenprobleme zwischen der Bahnregelung und der Bahnplanung frühzeitig behoben werden. Da sich im Laufe des Projektes gezeigt hat, dass die Fertigungsplanung den Gesamtsystemtesttermin nicht einhalten kann, wird der UDP-Dummy in Zusammenarbeit mit der Bahnregelung dahingehend erweitert, zufallsgenerierte Aufträge zu senden. Im letzten Schritt wird der UDP-Dummy als Fertigungsplanungseratz erweitert indem sich die Positionen der Werkstücke gemerkt werden und die zufalls generierten Aufträge auf ausführbare Aufträge gefiltert wird. Zum Ende des Projektes wird zusätzlich eine Simulation, wie zu Beginn, mit der Erweiterung auf fünf Robotinos ausgeführt, um geringe Änderungen im Programm aufzuzeichnen, da die Aufzeichnung von Daten von mehreren Robotinos unter Simulink-Realtime einige Probleme verursacht. Unter Kapitel 5.1 wird eine Simulation des Ausweichmanövers dargestellt und unter Kapitel 5.2 wird das Verhalten beim Anfahren der Wartebereiche gezeigt.

## 5.1 Simulation des Ausweichmanövers

In diesem Abschnitt wird die Simulation des Ausweichmanövers dargestellt. Dazu werden zwei Simulationen miteinander verglichen. In der ersten Simulation, welche in Abbildung 5.1 dargestellt ist, werden zwei Robotinos betrachtet. Robotino 1, in Rot dargestellt, fährt von

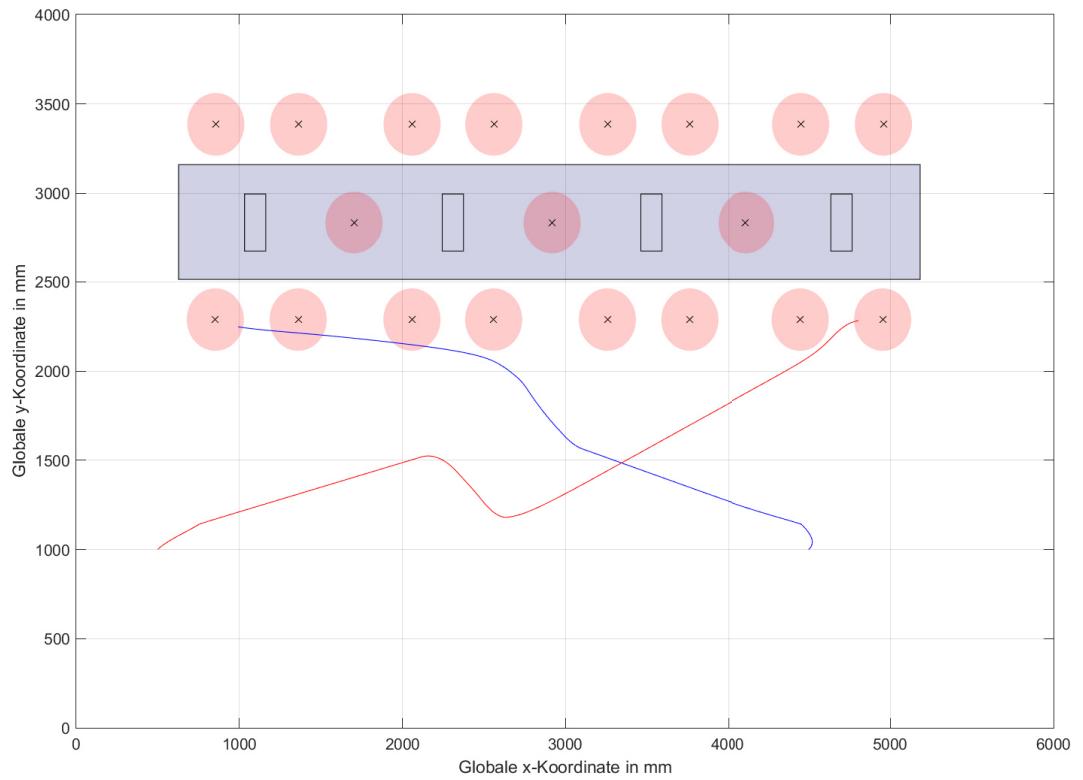


Abbildung 5.1: Simulation mit Ausweichfunktion

der Position [500 1000] zur Station 7, welche sich ganz rechts befindet. Robotino 2, in Blau dargestellt, fährt von der Position [4500 1000] zur Station 0, welche sich ganz links befindet. Wie in der Abbildung zu sehen ist, umfahren sich die Robotinos rechts herum.

In der zweiten Simulation, welche in Abbildung 5.2 dargestellt ist, werden die gleichen Robotinos mit der gleichen Start und Zielposition betrachtet. In dieser Simulation wird lediglich die Ausweichfunktion auf Null gesetzt. Wie in der Abbildung zu erkennen ist, kommt es zu einem Abstoßverhalten, wodurch die Robotinos stark ausgebremst werden. Wie in den Abbildungen zu erkennen ist, hat die entwickelte Ausweichfunktion eine Verbesserung im Ausweichverhalten verursacht.

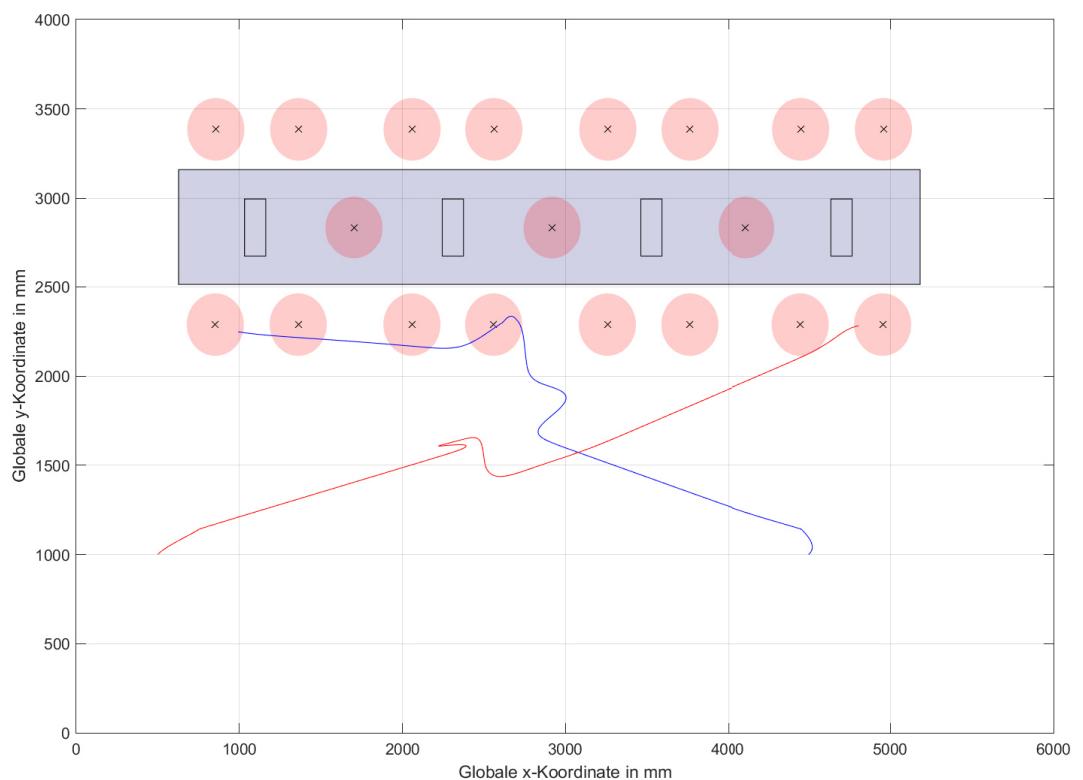


Abbildung 5.2: Simulation ohne Ausweichfunktion

## 5.2 Simulation Wartebereiche

In diesem Abschnitt wird eine Simulation der Wartebereiche durchgeführt. Dazu werden vier Robotinos im Transportbereich verteilt und sollen das selbe Ziel anfahren. Dadurch kann man das Verhalten der Robotinos bei der Nutzung der Wartepositionen beobachten. In Abbildung 5.3 ist das Ergebnis der Simulation dargestellt. Die Zielstation wird in dieser Simulation als 2 definiert. Dabei ist zu erkennen, dass der Robotino mit der Startposition [2000 1000], in Grün dargestellt, als erster das Ziel erreicht.

Wie am nächsten Robotino, in Blau dargestellt, zu erkennen ist, wird von seiner Startposition ([1000 500]) erst die Station als Ziel angenommen. Da sich in diesem Moment jedoch der grüne Robotino in seiner Fahrtrichtung befindet, wird das unter Kapitel 5.1 bereits simulierte Ausweichmanöver angewendet, bis der grüne Robotino die Station erreicht hat. Dieser Zeitpunkt ist in der Abbildung als schlagartige Richtungsänderung erkennbar, da in diesem Moment das Ziel auf die erste Warteposition geändert wird. Da sich auf dem Weg zum neuen Ziel keine Robotinos befinden, trifft der blaue Robotino auf der ersten Warteposition als erster ein.

Der rote Robotino, mit der Startposition [5000 1000], zeigt bei seiner Fahrt ein ähnliches Verhalten wie der blaue. An der Route kann erkannt werden, dass zuerst gerade auf die Station zugefahren wird. In nächstem Schritt wird schlagartig die Richtung auf die erste Warteposition geändert, da der grüne Robotino die Station erreicht hat. Die nächste Auffälligkeit besteht darin, dass das Ausweichmanöver aufgrund des blauen Robotinos ausgelöst wird. Dadurch wird der Robotino von seiner Ideallinie abgebracht. Da sich der rote Robotino, aufgrund des Ausweichmanövers, beim Eintreffen des blauen Robotinos, auf der ersten Warteposition auf der linken Seite der ersten Warteposition befindet, wird der auf der ersten Warteposition befindliche Robotino umfahren, da sich die Wartepositionen bei geraden Zielstationen nach rechts anhängen.

Als letztes wird der cyane Robotino betrachtet. Da dieser Robotino über den Stationen ([1000 3500]) startet, wird dieser zuerst über die Führungspotentialfelder zur Hauptfahrzone transportiert. Da sich der grüne Robotino beim Wechsel zur Hauptfahrzone bereits in der Station befindet, wird direkt die erste Warteposition als Ziel definiert. Da dieser Robotino auf seiner Route auf den roten Robotino trifft, wird auch hier im Laufe der Route ein Ausweichmanöver erkennbar. Da sich auch dieser Robotino auf die nächsten Wartepositionen begeben möchte, sich jedoch der rote Robotino immer vor ihm befindet, stellt sich der cyane Robotino im Laufe der Simulation auf die dritte Warteposition.

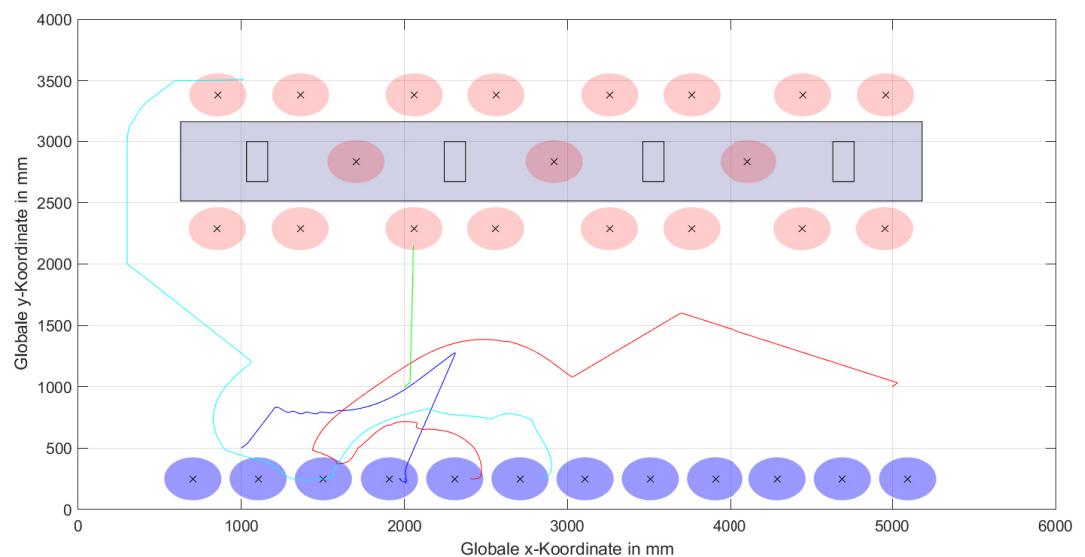


Abbildung 5.3: Simulation der Wartebereiche

# 6 Implementierung und Zielsysteme

In diesem Kapitel wird die Umsetzung der einzelnen Algorithmen des in Kapitel 4 erläuterten Konzepts mit Hilfe der Software Matlab/Simulink erklärt. Für einen besseren Überblick zu dem Konzept zeigt die Abbildung 6.1 ein Ablaufdiagramm des Programms der Bahnplanung.

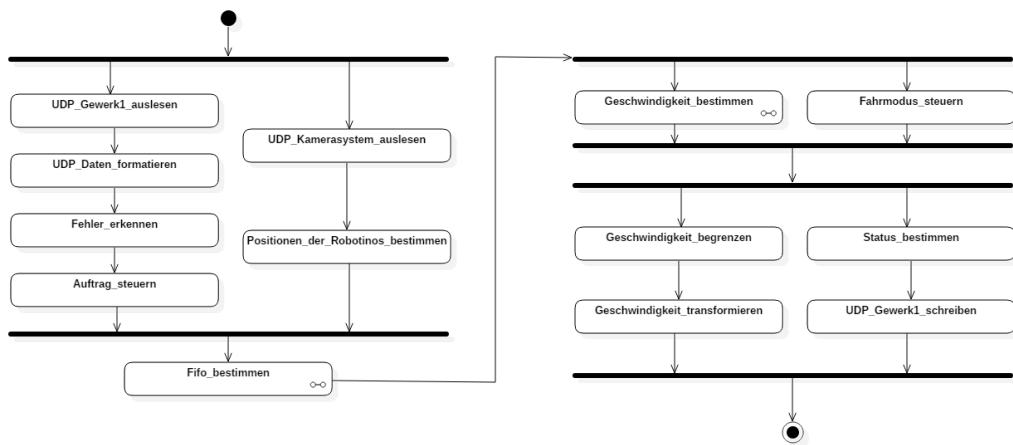


Abbildung 6.1: Ablaufdiagramm Konzept

Zunächst wird der Auftrag der Fertigungsplanung verarbeitet und parallel dazu wird die eigene Position des Robotinos sowie die Positionsdaten der anderen Robotinos bezogen. Nachdem die Selbstorganisation durch Wartebereiche abgearbeitet wurde, wird das erhaltene Ziel weitergeleitet. Mit dem bekannten Ziel wird der Zielvektor bestimmt. Die Hinderniserkennung läuft nun gleichzeitig. Hat der Robotino sein Endziel erreicht, kann ein neuer Auftrag abgearbeitet werden.

Hat ein Robotino zur Zeit keinen Auftrag, wird als Ziel die Standby-Position einprogrammiert. Zunächst wird der Grundalgorithmus zur Zielgebung erläutert und anschließend um die einzelnen Unterpunkte des Konzeptes erweitert.

## 6.1 Selbstorganisation durch Wartebereiche

Die in dem Unterkapitel 4.2.1 erklärte intelligente Selbstorganisation wird im Folgenden erklärt und es wird gezeigt, wie der Algorithmus in der Software Simulink umgesetzt wird.

Das Ablaufdiagramm in Abbildung 6.2 stellt den Ablauf des Algorithmus dar.

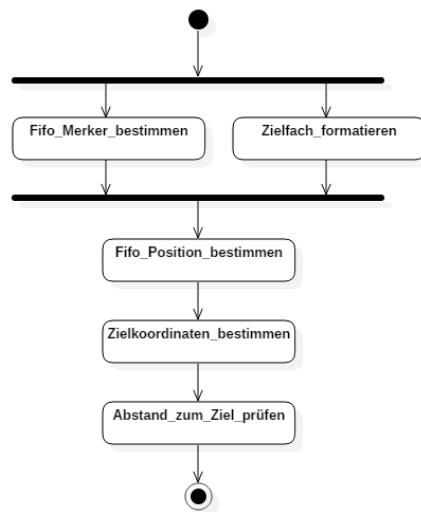


Abbildung 6.2: Ablaufdiagramm der Selbstorganisation

Die Positionen der anderen Robotinos werden über den 'FIFO Merker' geprüft und es wird geschaut, ob die anderen Roboter Positionen einnehmen, die ein zugehöriger Warteplatz für die Zielstation darstellen.

Die Roboterpositionen werden in einem Array der X-Positionen und einem der Y-Positionen gespeichert. Die 12 bestehenden in gleichmäßigen Abstand aufgeteilten Wartebereiche können mit der Formel 6.1 in Zeile 8 des Matlab-Codes 6.1 in X-Richtung berechnet werden.

$$N(n) = \text{round}((X(n) - 703)/400) + 1; \quad (6.1)$$

Stimmen die X-Positionen der Robotinos mit den Bereichen überein und sind die Y-Positionen am linken Rand des gesamten 'Fabrik'-Bereiches, wird der Wartebereich an dem sich ein Robotino befindet, mit einer jedem Robotino spezifischen Nummer belegt. Damit die Wartebereiche auch in jedem Zyklus wieder neu berechnet werden, wird der

Merkern zunächst auf null gesetzt und in jedem Zyklus mit neuen Werten beschrieben.

Listing 6.1: FIFO Merker

```
function [FifoArray1, StationArray1, LadeArray1] = fcn(Robbi, FifoArray, LadeArray)
X=[Robbi(1) Robbi(4) Robbi(7) Robbi(10)];
Y=[Robbi(2) Robbi(5) Robbi(8) Robbi(11)];
FifoArray1=zeros([1 12]);
FifoArray1=FifoArray(1:12);
N=zeros([1 5]);
for n= 1 : 1 : 4
    N(n)=round((X(n)-703)/400)+1; % Berechnung in welchem Fifo der Roboter sich befindet
elseif (Y(n)<450 && (X(n)>100 && X(n)<5250)&& N(n)<=12 && N(n)>0)
    for i=1:12
        if FifoArray1(i)==n
            FifoArray1(i)=0;
        end
    end
    FifoArray1(N(n))=n;
    ...
    ...
end
```

Auch die Belegung der Stationen wird mit dem selben Algorithmus berechnet. Einzig die Bereichsabfrage ändert sich und die Formel 6.1 für die Berechnung der Belegung. Hierbei ändert sich allerdings nur die Unterteilung und der Offset der Formel.

Die Merkerwerte werden in einem Stateflowdiagramm weiterverarbeitet. Die Eingänge des Stateflowdiagramms sind die Auftragsdaten, die Merkerdaten, und die genauen Positionsdaten des Robotinos aus Beobachter und Kamerawerten. Damit das System die Wartepositionen der zugehörigen Zielstationen für die Zuweisung verwendet, wird die Formel 6.2 benutzt. Die zwölf Wartebereiche können so explizit den acht Stationen zugeordnet werden. Zwei Stationen teilen sich jeweils drei Wartebereiche.

$$\text{ersterFifo} = \text{floor}(\text{Zielstation}/2) \cdot 3 + 2 \cdot \text{mod}(\text{Zielstation}, 2) + 1 \quad (6.2)$$

$$\text{zweiterFifo} = \text{floor}(\text{Zielstation}/2) \cdot 3 + 1 + 1 \quad (6.3)$$

$$\text{dritterFifo} = \text{floor}(\text{Zielstation}/2) \cdot 3 + 2 - 2 \cdot \text{mod}(\text{Zielstation}, 2) + 1 \quad (6.4)$$

Ist der erste Bereich belegt wird der zweite Bereich abgefragt ist dieser belegt wird der Dritte abgefragt. Sind alle drei Bereich belegt, gibt das Stateflowdiagramm den höchsten Wert aus. Beim höchsten Wert wird die Standbyposition als Ziel einprogrammiert.

Wird im nächsten Zyklus der nächst kleinere Wartebereich frei, so wird dieses als Ziel angefahren. In dem Fall, wenn ein anderer Robotino den ersten Wartebereich verlässt und zur Zielstation fährt, besteht ein Sonderfall. Da die Strecke zwischen den beiden Punkten ca. zwei Meter beträgt dauert diese Fahrt mehrere Zyklen. Damit ein dahinter stehender Robotino in dem Moment nicht auch direkt die Station anfährt, weil alle anderen Bereiche frei sind, wird eine weitere Position abgefragt. Die Stationsbelegung wird abgefragt. Die Station wird erst wieder freigegeben, wenn sie im Stationsmerker nicht mehr belegt ist.

## 6.2 Bekannte Hindernisse

Für eine verbesserte Kollisionsvermeidung wurde das Ausweichmanöver, wie in Unterkapitel 4.2.2 bereits erklärt, verwendet. Die Aufgabe des Manövers ist es den Zielvektor des Robotinos so zu verändern, dass er einem sich im Weg befindenden Robotino mit einem größeren Abstand ausweicht. Es werden die Position des Robotinos benötigt sowie die Positionen der anderen Robotinos, die Störpositionen. Darüber hinaus benötigt das Manöver den Zielvektor.

Zunächst wird analysiert in welchem Winkel die Störpositionen sich zum Zielvektor befinden. Liegt ein anderer Robotino innerhalb von  $30^\circ$  zum Zielvektor wird ein Ausweichwinkel berechnet.

$$\text{Ausweich}(1, i) = +\text{asin}(\text{Abstand}(2, i)/\text{Hyp}) - \pi/2 \quad (6.5)$$

Der erste Wert des Ausweichs-Array aus Formel 6.5 ist der Ausweichwinkel. Über eine trigonometrische Funktion wird der genaue Winkel des Zielvektors zum Ziel ausgerechnet und  $90^\circ$  addiert. Solange eine Störposition innerhalb der  $30^\circ$  liegen, bleibt der Ausweichwinkel erhalten. Ist der im Weg liegende Roboter weit genug umfahren, wird der originale Zielvektor einprogrammiert.

## 6.3 Unbekannte Hindernisse

Unbekannte Hindernisse werden nur mit Hilfe der neun verbauten Infrarotsensoren erkannt. Zyklisch werden diese Sensoren abgefragt. Detektiert einer der Sensoren ein Hindernis wird zunächst überprüft, an welcher Position relativ zur Roboterausrichtung sich das Hindernis befindet. Der Ablauf wird im Code 6.2 dargestellt:

Listing 6.2: Infraroterkennung

```
for x=1:9
    if (isnan(IRDaten(x))==0) && (IRDaten(x) <= 150)
        IRWerte(1,x)=cos (((x-1)*40*pi/180)+RobPos(3))*(IRDaten(x)+200)+RobPos(1);
        IRWerte(2,x)=sin (((x-1)*40*pi/180)+RobPos(3))*(IRDaten(x)+200)+RobPos(2);
    else
        IRWerte(1,x)=NaN;
        IRWerte(2,x)=NaN;
    end
end
```

Wurde ein Hindernis detektiert, wird das Hindernis in globalen Koordinaten bestimmt. Diese berechneten 'IRWerte' werden anschließend in der Zielvektorberechnung aus Kapitel 6.4 einbezogen.

## 6.4 Potentialfeld

Um das unter Kapitel 4.1 beschriebene Potentialfeldkonzept in Matlab/Simulink umzusetzen, werden die Funktionen (??) bis (??) partiell nach x und y abgeleitet. Die dabei entstehenden Funktionen sind als (6.6) bis (6.25) definiert. Durch die Ableitung des Potentialfeldes ergibt sich der Gradient, der negiert werden muss, damit der ermittelte Vektor Richtung Ziel zeigt.

$$Vx_{Hindernis}(x, y) = -K \cdot e^{-\frac{(x - Stationsposition_x)^2 + (y - Stationsposition_y)^2}{2 \cdot \sigma^2}} \cdot \frac{x - Stationsposition_x}{2 \cdot \sigma^2} \quad (6.6)$$

$$Vy_{Hindernis}(x, y) = -K \cdot e^{-\frac{(x - Stationsposition_x)^2 + (y - Stationsposition_y)^2}{2 \cdot \sigma^2}} \cdot \frac{y - Stationsposition_y}{2 \cdot \sigma^2} \quad (6.7)$$

$$Vx_{Ziel}(x, y) = \frac{K \cdot (x - Ziel_x)}{\sqrt{(x - Ziel_x)^2 + (y - Ziel_y)^2}} \quad (6.8)$$

$$Vy_{Ziel}(x, y) = \frac{K \cdot (y - Ziel_y)}{\sqrt{(x - Ziel_x)^2 + (y - Ziel_y)^2}} \quad (6.9)$$

$$Vx_{Umwelt(Zone=0,x=0..5500,y=0..2500)}(x, y) = \frac{1}{\sigma} \cdot K \cdot e^{\frac{-x}{\sigma}} + \frac{-1}{\sigma} \cdot K \cdot e^{\frac{x-x_{max}}{\sigma}} \quad (6.10)$$

$$Vy_{Umwelt(Zone=0,x=0..5500,y=0..2500)}(x, y) = \frac{-1}{\sigma} \cdot K \cdot e^{\frac{y-2500}{\sigma}} + \frac{1}{\sigma} \cdot K \cdot e^{\frac{-y}{\sigma}} \quad (6.11)$$

$$Vx_{Umwelt(Zone=0,x=5500..x_{max},y=0..2500)}(x, y) = \frac{1}{\sigma} \cdot K \cdot e^{\frac{-(x-5500)+(y-2500)}{\sigma}} + \frac{-1}{\sigma} \cdot K \cdot e^{\frac{x-x_{max}}{\sigma}} \quad (6.12)$$

$$Vy_{Umwelt(Zone=0,x=5500..x_{max},y=0..2500)}(x, y) = \frac{-1}{\sigma} \cdot K \cdot e^{\frac{-(x-5500)+(y-2500)}{\sigma}} + \frac{1}{\sigma} \cdot K \cdot e^{\frac{-y}{\sigma}} \quad (6.13)$$

$$Vx_{Umwelt(Zone=0,x=5500..x_{max},y=2500..y_{max})}(x, y) = \frac{1}{\sigma} \cdot K \cdot e^{\frac{-x-5500}{\sigma}} + \frac{-1}{\sigma} \cdot K \cdot e^{\frac{x-x_{max}}{\sigma}} \quad (6.14)$$

$$Vy_{Umwelt(Zone=0,x=5500..x_{max},y=2500..y_{max})}(x, y) = \frac{-1}{\sigma} \cdot K \cdot e^{\frac{y-y_{max}}{\sigma}} + \frac{1}{\sigma} \cdot K \cdot e^{\frac{-y}{\sigma}} \quad (6.15)$$

$$Vx_{Umwelt(Zone=1)}(x, y) = K_{gerade} \quad (6.16)$$

$$Vy_{Umwelt(Zone=1)}(x, y) = -2 \cdot \frac{y - 3500}{K_{parabel}} \quad (6.17)$$

$$Vx_{Umwelt(Zone=2)}(x, y) = -K_{gerade} \quad (6.18)$$

$$Vy_{Umwelt(Zone=2)}(x, y) = -2 \cdot \frac{y - 3500}{K_{parabel}} \quad (6.19)$$

$$Vx_{Umwelt(Zone=3)}(x, y) = -2 \cdot \frac{x - 300}{K_{parabel}} \quad (6.20)$$

$$Vy_{Umwelt(Zone=3)}(x, y) = -K_{gerade} \quad (6.21)$$

$$Vx_{Umwelt(Zone=4)}(x, y) = 0 \quad (6.22)$$

$$Vy_{Umwelt(Zone=4)}(x, y) = K_{gerade} \quad (6.23)$$

$$Vx_{Umwelt(Zone=5)}(x, y) = -2 \cdot \frac{x - 5500}{K_{parabel}} \quad (6.24)$$

$$Vy_{Umwelt(Zone=5)}(x, y) = -K_{gerade} \quad (6.25)$$

Der Ablaufplan in Abbildung 6.3 zeigt den Ablauf zum ermitteln des Geschwindigkeitsvektors. Dazu werden zuerst alle benötigten Daten bestimmt. Dazu gehören die Positionen der bekannten Hindernisse, welche unter Kapitel 6.2 beschrieben sind, und der unbekannten Hindernisse, welche unter Kapitel 6.3 erläutert werden. Des Weiteren werden die von der Bahnregelung erhaltenen Beobachterdaten als Robotinoposition genutzt, welche in den Formeln die x und y Koordinaten darstellen. Da zuerst die Sollfahrtrichtung bestimmt werden muss, um ein Ausweichmanöver bestimmen zu können, wird erst der Geschwindigkeitsvektor anhand der Umwelt, des Ziels und der Hindernisse bestimmt. Im nächsten Schritt wird dann anhand der bekannten Hindernisse ein Ausweichmanöver berechnet, wenn sich ein bekanntes Hindernis in einem Toleranzband in Fahrtrichtung befindet. Im Anschluss wird das Ausweichmanöver auf den Geschwindigkeitsvektor addiert. Bevor der Geschwindigkeitsvektor auf den Sollwerteingang des Robotinos gegeben werden kann, muss der ermittelte Geschwindigkeitsvektor gefiltert und auf lokale Koordinaten transformiert werden.

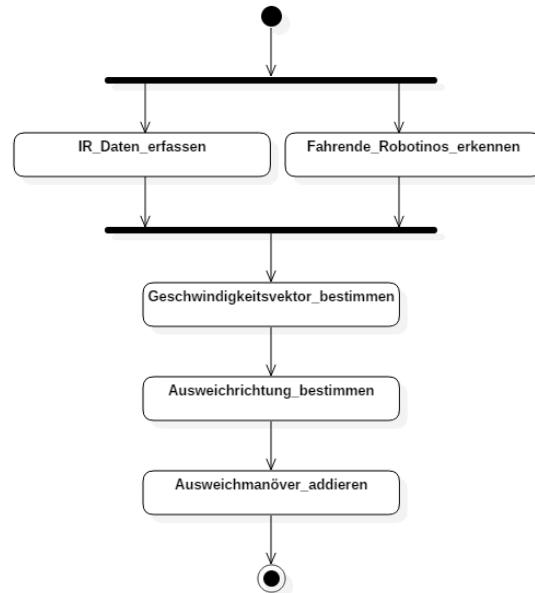


Abbildung 6.3: Ablaufplan des Subsystems Vektorberechnung

## 6.5 Funktionsblockbeschreibungen

In diesem Kapitel werden die Funktion der erstellten Funktionsblöcke und Stateflows erläutert.

### 6.5.1 UDP Gewerk1 Kommunikation

In diesem Subsystem wird die UDP-Kommunikation mit der Fertigungsplanung realisiert, dazu werden die zu sendenden Daten gepackt und per UDP an alle im Netzwerk geschickt. Des Weiteren werden die von der Fertigungsplanung gesendeten Daten ausgelesen. Der Aufbau des Subsystems ist in Abbildung 6.4 dargestellt. Um die UDP Kommunikation zu sichern, wird für jeden Robotino ein eigener sende und lese Port definiert. Dazu wird für den sende Port das Muster 250X0 verwendet und für den auslese Port das Muster 2591X. Dabei wird X als Robotinoid gesetzt. Da in Matlab der Winkel in Rad genutzt wird, wird vor dem Senden der Robotinoausrichtung der Winkel in Grad umgerechnet. Die einzulesenden und zusendenden Daten sind in Kapitel 4.3.2 definiert.

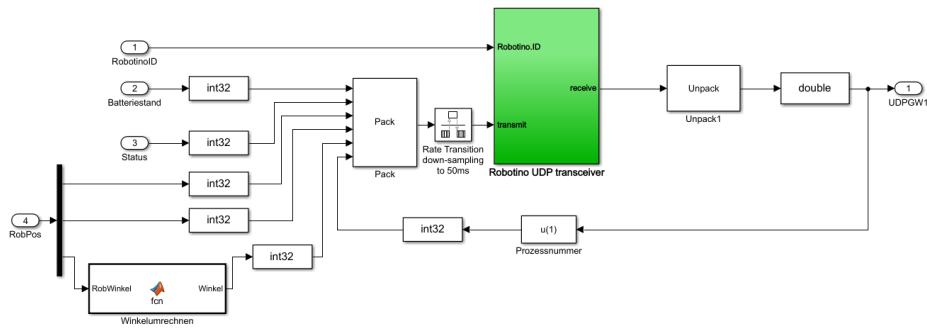


Abbildung 6.4: Subsystem UDP Gewerk 1

### 6.5.2 UDP Daten formatieren

In diesem Subsystem werden die eingelesenen UDP Daten von der Fertigungsplanung aufgesplittet. Der Aufbau des Subsystems ist in Abbildung 6.5 dargestellt. Des Weiteren wird erfasst, ob ein neuer Auftrag vorhanden ist und die Fach- und Stationsbeschreibung wird auf das im Programm genutzte Format angepasst.

### 6.5.3 Fehler erkennen

In dieser Funktion wird sich der Fehlerstatus des Robotinos gemerkt und bei einem Reset-Signal durch die Fertigungsplanung zurückgesetzt. Des Weiteren wird der Fehler erkannt, wenn ein Werkstück während des Transport verloren wird.

### 6.5.4 Auftrag steuern

In diesem Stateflow wird das abarbeiten des Auftrags gesteuert, da wir von der Fertigungsplanung Aufträge gesendet bekommen, die Startfach und Zielfach enthalten, ist dies notwendig. Das entwickelte Stateflow ist in Abbildung 6.6 dargestellt. Dabei ist der Startzustand auf einen neuen Auftrag warten. In diesem Zustand wird der Status auf „Frei“ gesetzt. Wenn nun ein Auftrag eingegangen ist, wird unterschieden ob der Robotino laden oder transportieren soll. Wenn der Robotino laden soll, wird als Ziel die Ladestation an die folgenden Funktionen übergeben. Wenn nun ein Signal von der Fertigungsplanung erhalten wird, dass der Robotino fertig geladen hat, wird der Status auf „Ladenabgeschlossen“ gesetzt. Wenn sich nun der Robotino im Transportbereich befindet, also die Ladestation sicher verlassen wurde, wird wieder auf einen neuen Auftrag gewartet. Wenn ein Auftrag erhalten wird, der zum Transport dient, wird der Zustand auf „Arbeitend ohne Werkstück“ geändert. In diesem Zustand wird

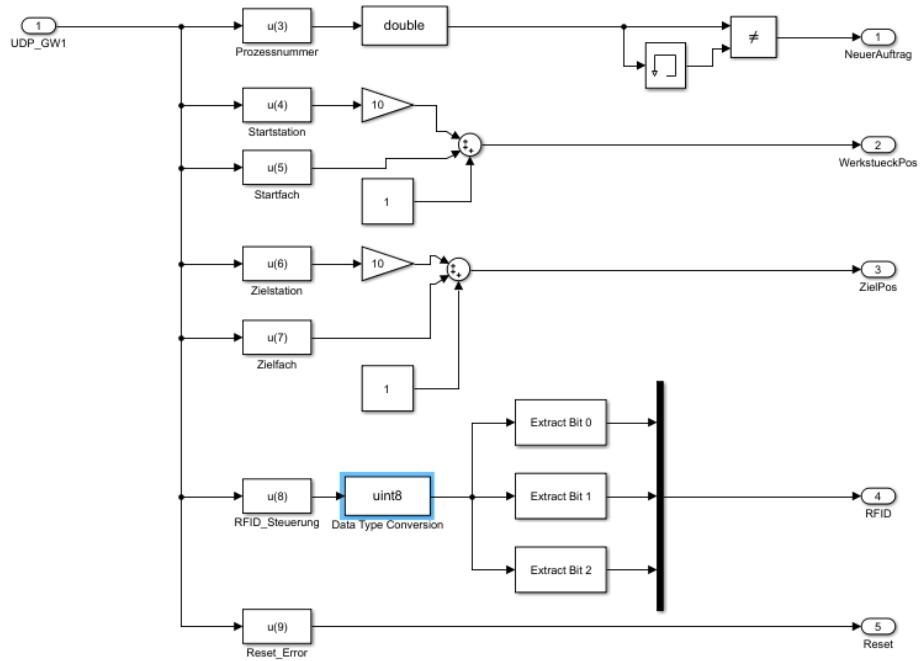


Abbildung 6.5: Subsystem UDP Daten formatieren

als Ziel das Startfach an die nachfolgenden Funktionen übergeben. Wenn nun ein Werkstück aufgenommen ist und sich der Robotino wieder im Transportbereich befindet, wird der Zustand auf „Arbeitend mit Werkstück“ gesetzt. In diesem Zustand wird nun das Zielfach als Ziel an die nachfolgenden Funktionen übergeben. Wenn dann das Werkstück abgelegt ist, wird der Zustand auf Werkstück abgelegt geändert. Dieser Zustand wird verlassen, wenn der Robotino wieder im Transportbereich ist und es kann wieder ein neuer Auftrag begonnen werden. Des Weiteren wird in diesem Stateflow ein Teil des Fehlerhandlings durchgeführt. Dazu gehört zum Beispiel, dass der Robotino seinen Status auf „auf einen neuen Auftrag warten“ setzt, wenn das Werkstück während des Transportes verloren geht oder kein Werkstück am Startfach aufgenommen werden kann.

### 6.5.5 Position der Robotinos bestimmen

In dieser Funktion werden anhand der RobotinOID und der UDP Daten des Kamerasystems die Störposition (andere Robotinos) bestimmt sowie die eigene Position bestimmt. Da im Laufe des Projektes Beobachterdaten durch die Bahnregelung generiert werden, wird die eigene Position nach dieser Funktion mit den Beobachterdaten überschrieben.

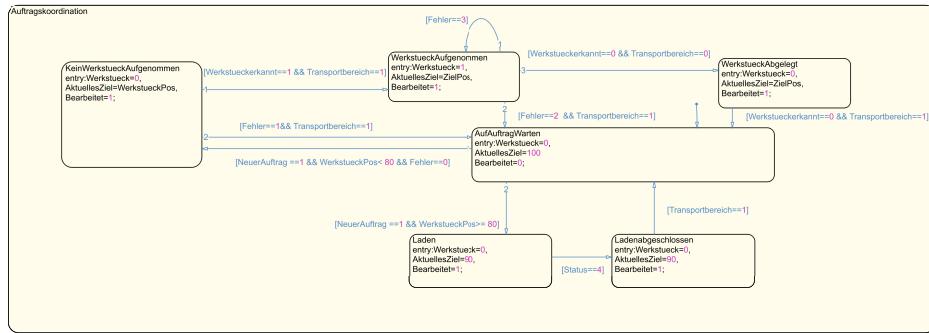


Abbildung 6.6: Stateflow Auftrag steuern

### 6.5.6 Fifo bestimmen

In diesem Abschnitt wird beschrieben welche Aufgaben die zur Wartepositionsbestimmung Funktionen erfüllen.

#### Fifo Merker bestimmen

In dieser Funktion werden globale Merker für die Wartepositionen und Stationen bestimmt. Dabei werden die Stationsmerker nur gesetzt, wenn sich ein Robotino in der Station befindet und zurückgesetzt, wenn die Station sicher verlassen wurde. Des Weiteren werden die Wartepositionsmerker gesetzt, wenn sich ein Robotino auf der Warteposition befindet. Im Gegensatz zu den Stationsmerkern wird der Wartebereichsmerker erst zurückgesetzt, wenn sicher ein neuer Wartebereich oder die Station erreicht wurde.

#### Zielfach formatieren

In dieser Funktion wird das aktuelle Ziel in Zielstation und Zielfach aufgeteilt, da in dem folgenden Stateflow hauptsächlich mit der Stationsnummer gerechnet wird.

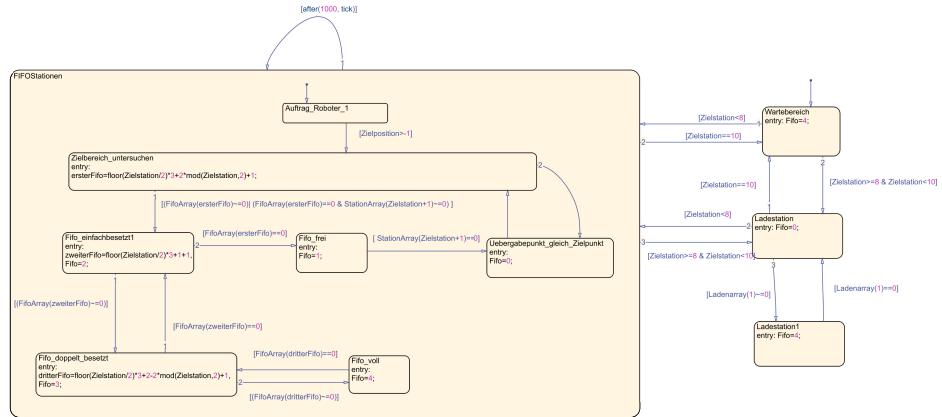


Abbildung 6.7: Stateflow Wartebereich bestimmen

### Fifo Position bestimmen

In diesem Stateflow, welches in Abbildung 6.7 dargestellt ist, wird die Warteposition des Robotinos bestimmt. Dazu werden die in Kapitel 6.1 beschriebenen Formeln genutzt. Dabei wird die Warteposition 4 als globale Warteposition definiert. Wenn der Robotino kein Ziel hat, wird die globale Warteposition als Ziel definiert. Sobald ein Ziel angefahren werden soll, wird zuerst die erste Warteposition abgefragt. Wenn die erste Warteposition belegt ist, wird die zweite Warteposition abgefragt. Dies geschieht solange bis Warteposition 4 erreicht wird. Wenn die erste Warteposition nicht belegt ist, wird die Station abgefragt. Ist die Station belegt, wird die Warteposition auf 1 gesetzt. Im Gegensatz dazu wird die Station als Ziel gesetzt, wenn diese nicht belegt ist. Ein Sonderfall in dieser Betrachtung ist die Ladestation. Dabei wird direkt die Ladestation abgefragt. Wenn die Ladestation belegt ist, wird die globale Warteposition angefahren, sonst wird die Ladestation als Ziel festgelegt.

### Zielkoordinaten bestimmen

In dieser Funktion wird anhand der ermittelten Warteposition die Zielkoordinaten berechnet, wenn das Ziel eine Warteposition ist. Wenn das Ziel die globale Warteposition ist, wird die Zielkoordinaten anhand eines definierten Vektors ausgelesen. Des Weiteren werden die

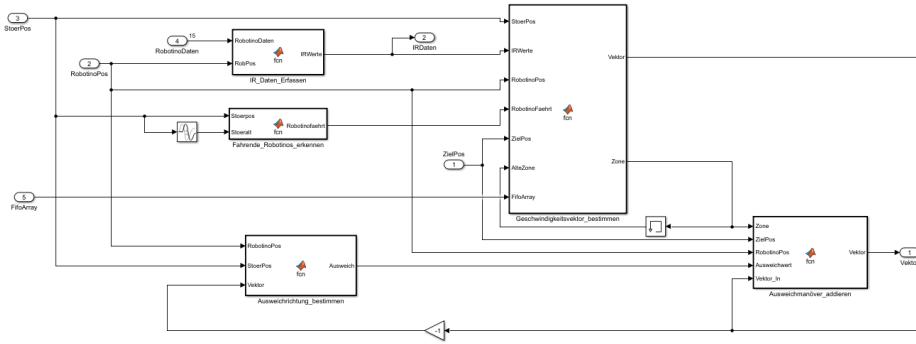


Abbildung 6.8: Subsystem Vektorberechnung

Zielkoordinaten aus einem anderen Vektor bestimmt, wenn die Station angefahren werden soll.

### Abstand zum Ziel prüfen

In dieser Funktion wird geprüft, ob die Station erreicht ist. Diese Information wird im späteren Programmverlauf dazu genutzt vom Transportbereich zum Fertigungsbereich zu wechseln.

#### 6.5.7 Geschwindigkeit bestimmen

Um die Sollgeschwindigkeit des Robotinos zu bestimmen, wird das Subsystem „Vektorberechnung“ genutzt. Dieses Subsystem ist wie in Abbildung 6.8 dargestellt, aufgebaut. Das Subsystem hat dabei den in Abbildung 6.3 gezeigten Ablauf. Dabei ist zu erkennen, dass zuerst die IR-Daten erfasst und die fahrenden Robotinos erkannt werden. Die IR-Datenerfassung wird unter Kapitel 6.5.7 näher erläutert. Wie die fahrenden Robotinos erkannt werden, wird in Kapitel 6.5.7 beschrieben. Anhand der IR-Daten und der Informationen über die fahrenden Robotinos wird der Geschwindigkeitsvektor bestimmt. Die dabei genutzte Funktion wird unter Kapitel 6.5.7 beschrieben. Anhand des bestimmten Geschwindigkeitsvektors wird eine Ausweichrichtung bestimmt, wenn ein anderer Robotino die Route kreuzt. Die dazu genutzte Funktion ist in Kapitel 6.5.7 näher beschrieben. Im nächsten Schritt wird dann das Ausweichmanöver ausgeführt. Die Implementierung des Ausweichmanövers wird in Kapitel 6.5.7 beschrieben.

### IR Daten erfassen

In der IR-Daten erfassen Funktion werden zuerst die IR-Daten aus den RobotinoDaten gefiltert und auf NaN gesetzt, wenn der gemessene Abstand größer als 150 mm ist. Bei einem Abstand kleiner gleich 150 mm wird die Position der Hindernisse in globalen Koordinaten bestimmt und ausgegeben.

### Fahrende Robotinos erkennen

Um zu erkennen, ob die anderen Robotinos fahren, werden die Postionen der Robotinos mit den Postionen vor einer Sekunde verglichen. Wenn die Robotinos mehr als 10 mm in der Sekunde zurückgelegt haben, wird der Robotino als fahrend gesetzt.

### Geschwindigkeitsvektor bestimmen

In dieser Funktion wird das in Kapitel 6.4 beschriebene Potentialfeld zur Vektorberechnung umgesetzt. Die genutzten Ableitungen sind unter (6.6) bis (6.25) dargestellt. In der Funktion (6.6) und (6.7) wird die Ableitung des Stationspotentialfeldes definiert. Diese Funktionen werden im Programm für die vier Stationen und die zwei Ladestation ausgeführt. Im nächsten Schritt wird der Zielvektor über die abgeleitete Funktion (6.8) und (6.9) bestimmt. Im nächsten Schritt werden die Robotino- und IR-Gradienten berechnet. Dazu wird die gleiche Funktion ((6.6) und (6.7)) wie bei der Stationsberechnung genutzt. Dabei werden jedoch die über die IR-Daten erfassten Positionen, sowie die per Kamera-UDP erhaltenen Robotinopositionen genutzt. Im nächsten Schritt wird die aktuelle Robotinozone bestimmt und je nach Zone der Umweltgradient, nach den Funktionen (6.10) bis (6.25), bestimmt. Da nicht alle Gradienten in allen Zonen genutzt werden, gibt es in jeder Zone eine eigene Summe von Gradienten. Dabei wird zum Beispiel die Stationsgradienten der Stationen nur in Zone 4 und die Ladestationsgradienten nur in Zone 0 mit eingerechnet. Des Weiteren wird der Zielgradienten nur in Zone 0 eingerechnet damit die Führung des Robotinos hinter den Stationen nicht gestört wird.

### Ausweichrichtung bestimmen

Um eine Ausweichrichtung zu bestimmen, wird zuerst der Abstand zu den anderen Robotinos anhand derer Positionen bestimmt. Wenn der Abstand geringer als ein Meter ist, wird der globale Winkel zum Robotino bestimmt und um  $90^\circ$  erhöht. Zusätzlich wird im nächsten Schritt der Fahrwinkel anhand des Geschwindigkeitsvektors in globalen Koordinaten bestimmt. Befindet sich ein Robotino in einem Toleranzbereich um den Fahrwinkel, so wird

mittels Fahr- und Robotinowinkel ein Enable-Bit gesetzt. Wenn dieses Enable-Bit nicht gesetzt ist, wird kein Ausweichmanöver ausgeführt.

### Ausweichmanöver addieren

In dieser Funktion wird nun auf das Ausweichmanöver der Geschwindigkeitsvektor addiert, wenn das Enable-Bit aus der Ausweichrichtungsbestimmung gesetzt ist. Des Weiteren wird das Ausweichmanöver nicht ausgeführt, wenn sich der Robotino nicht in der Hauptfahrzone (Zone=0), nicht im FIFO-Bereich oder nahe am Ziel (Distanz >300mm) befindet. Zusätzlich wird in dieser Funktion die Abbremsfilterung integriert, wie im Kapitel ?? beschrieben, wird dazu ein distanzabhängiger Faktor multipliziert.

### 6.5.8 Fahrmodus steuern

Das Stateflow, welches in Abbildung 6.9 dargestellt ist, steuert die Kommunikation mit der Bahnregelungsgruppe. Beim Start ist der Transportbereich, in dem die Potentialfeldmethode zum Regeln genutzt wird, aktiv. Wenn der Robotino über das Potentialfeld sein Ziel erreicht hat, wird an die Bahnregelung, über das setzen des Fertigungsbereich auf „Eins“, übergeben. Wenn das Ziel eine Ladestation ist, wird der Bahnregelung die Zielladestationsposition übergeben. Wenn der Robotino fertig geladen hat, wird der Bahnregelung wieder der Übergabepunkt als Ziel gegeben und wieder auf den Transportbereich gewechselt.

Wenn das Ziel eine Ladestation ist, werden zwei Varianten unterschieden. Die erste Möglichkeit besteht darin ein Werkstück aufzunehmen. Dabei wird zuerst die „Werkstueckposition“ übergeben. Nach der Bestätigung von der Bahnregelung des erfolgreichen Aufnehmens, wird der RFID-Leser als Ziel vorgegeben. Wenn die Bahnregelung die Position bestätigt, wird auf eine Bestätigung, des RFID-Schreibens durch die Fertigungsplanung gewartet. Nach dieser Bestätigung wird der Bahnregelung ein Übergabepunkt vorgegeben, an dem zur Potentialfeldmethode gewechselt werden soll. Dabei besteht die Möglichkeit nach hinten abzugeben, wenn ein Robotino auf die Station wartet, oder nach vorne, wenn kein Robotino in die Station möchte.

Im zweiten Fall soll ein Werkstück in der Station abgegeben werden. Dabei ändert sich die Reihenfolge der Ziele, so dass als erstes zum RFID-Leser und auf Bestätigung von Fertigungsplanung gewartet werden soll. Anschließend folgt die Zielposition für das Werkstück und wieder die Ausfahrposition.

In diesem Stateflow erfolgt zusätzlich die Statuszuweisung des Robotinos und dessen Fehlererkennung.

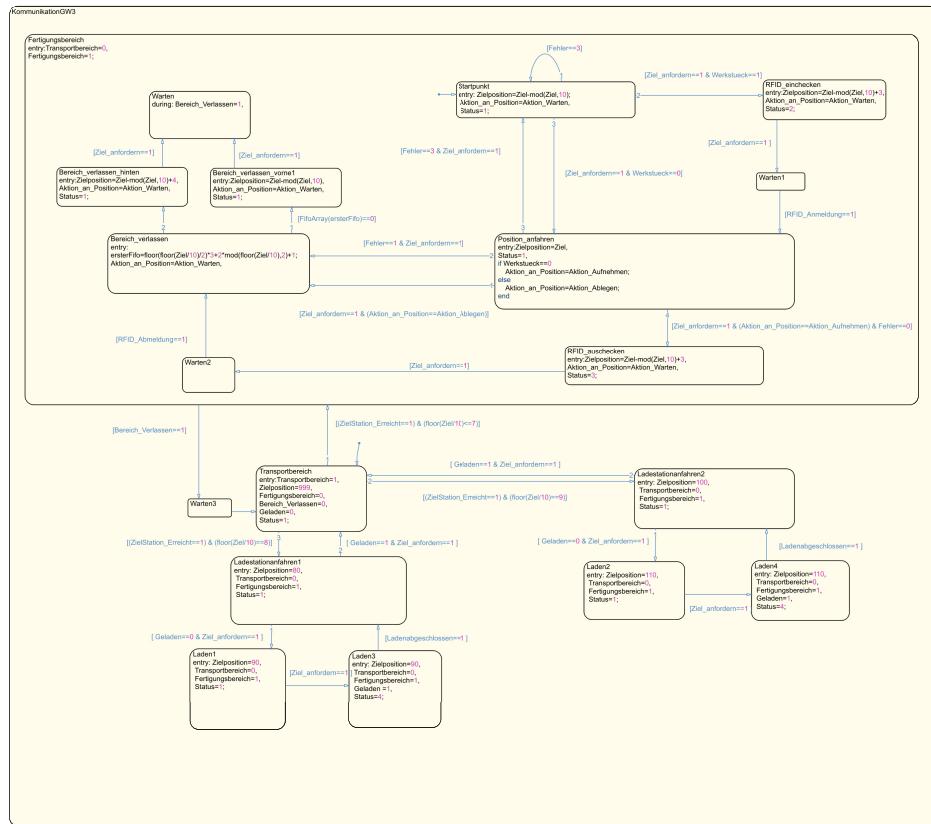


Abbildung 6.9: Fahrmodus steuern

### 6.5.9 Geschwindigkeit begrenzen

In dieser Funktion wird eine gleitende Mittelwertbildung über 100 Werte vorgenommen, um die Geschwindigkeitsvorgabe zu glätten. Nach der Filterung wird die Geschwindigkeit je nach Position begrenzt. Dabei wird die Geschwindigkeit bei den FIFO-Positionen auf 300mm/s und in der Hauptfahrzone auf 700mm/s begrenzt.

### 6.5.10 Geschwindigkeit transformieren

In dieser Funktion wird der globale Geschwindigkeitsvektor auf das Robotinokoordinaten-system über den Winkel des Robotinos transformiert. Dazu wird eine Rotationsmatrix genutzt. Da der Robotino den negativen Gradienten in mm/s als Vorgabe benötigt, wird der Geschwindigkeitsvektor vor diesem Funktionsblock mit -1000 multipliziert, da durch unsere Berechnung der positive Gradient in m/s ermittelt wird.

### 6.5.11 Status bestimmen

In dieser Funktion wird der Status des Robotinos, der an die Fertigungsplanung übergeben wird, bestimmt. Dabei wird der Status, der in der Fahrmodussteuerung unter Kapitel 6.5.8 bestimmt wird, direkt übergeben, solange kein Fehler durch die Bahnregelung registriert wird.

## 7 Robotino 2.0

In diesem Kapitel wird beschrieben welche Anpassungen am Programm vorgenommen werden müssen, um das Programm von den alten Robotinos auf dem neuen Robotino lauffähig zu bekommen. Um das Programm auf den Robotino 2.0 anzupassen wird eng mit der Bahnregelung und Gwerk4 zusammengearbeitet. Die Hauptänderung des Programmes besteht darin, den Simulinkblock zur UDP-Kommunikation durch ein Output und Input in den gesamt Bahnplanungsblock zu ersetzen, da die UDP-Kommunikationsblock von Simulink nicht in Twincat ausgeführt werden kann. Um in Twincat UDP-Daten zu schreiben oder zu lesen, müssen die per Input und Ouput definierten Daten auf den in Twincad integrierten UDP-Kontroller gemappt werden. Zusätzlich muss das Starterkit angepasst werden. Da das Programm unabhängig vom Starterkit aufgebaut ist, müssen keine großen Änderungen am Programm vorgenommen werden.

## 8 Validierung

Nachdem die Konzepterstellung abgeschlossen war, wurde mit der Implementierung der Software begonnen. Um bereits ganz zu Beginn die Software testen zu können, wurde ein Simulationsprogramm erstellt, auf dem das Potentialfeld abgebildet war und die Fahrten der Robotinos nachvollzogen werden konnten. Da die Gundfunktion in der Simulation einwandfrei funktionierte, wurde frühzeitig begonnen die einzelnen Softwareteile auf die Hardware der Robotinos zu übertragen und somit am realen System zu testen.

Über eine eigens programmierte Software als Dummy könnten über UDP-Kommunikation manuell Aufträge an die Robotinos gesendet werden. In Kooperation mit dem Gwerk3 konnten so vereinzelt Aufträge abgearbeitet werden. In diesen Testphasen wurden kleinere Fehler und Ungenauigkeiten erkannt und behoben. So führte beispielsweise das Anstellen in den Fifos zunächst zu Problemen, da die Potentialfelder zu groß waren und sich die Robotinos in Kombination mit den Infrarotsensoren gegenseitig aus der Warteschlange gedrängt haben.

Während der Testphase kam es zu Beginn häufiger zu Kollisionen mit dünnen Gegenständen, wie beispielsweise einem Stuhlbein. Befinden sich Hindernisse genau zwischen zwei IR-Sensoren, so werden diese nicht erkannt. Gelöst wurde dieses Problem zunächst mit einer konstanten Rotation der einzelnen Robotinos. Wie bereits beschrieben können die Robotinos sich erkennen und einander ausweichen. Ziel war es einen effizienten Produktionsablauf zu erschaffen. Durch die konstante Rotation der Robotinos verringert sich die Zuverlässigkeit des Ausweichmanövers. Nach ausgiebigem Testen wurde die Rotation wieder eingestellt. Die Priorität des Ausweichmanövers wird höher eingestuft, als die Erkennung schmaler Hindernisse.

Nach längerer Testfahrt mit aktiviertem Ausweichmanöver ist aufgefallen, dass es Bereiche gibt in denen ein Rechtsausweichen nicht zielführend ist. Bewegt sich ein Robotino in negativer X-Richtung ganz in der Nähe der Stationen und ein anderer Robotino liegt im Weg, so wird der Robotino durch die Ausweichfunktion in die Station 'gedrückt'. Durch die Randfunktionen des Transportfeldes wird der Robotino immer wieder abgestoßen. Das System gerät in Schwingungen. Aus diesem Grund wurde das Manöver insofern verbessert, dass kein Rechtsausweichen besteht, wenn der Robotino in negativer X-Richtung fährt und seine Entfernung in Y-Richtung zu den Stationen kleiner gleich ca. 50 cm beträgt.

Das Testen mit allen fünf Robotinos zur gleichen Zeit, war trotz festgelegtem Termin für einen

Gesamtsystemtest zunächst nicht möglich. Über ein Softwareprogramm, das zufällig und kontinuierlich Aufträge generiert, könnte schließlich ein Langzeittest durchgeführt werden. Bei den Langzeittest kam es in Situationen, in denen mehrere Robotinos aufeinander treffen und sehr wenig Platz zum Ausweichen war, zu vereinzelt Kollisionen. Diese Kollisionen traten vor allem dann auf, wenn sich in dem Konfigurationsraum alle fünf Robotinos bewegten. Die noch vorhandene Ungenauigkeit des Robotino 2.0, die besonders durch seine größeren Maße verursacht werden, und der begrenzte Konfigurationsraum erschweren einen komplett fehlerfreien Ablauf.

Durch die frühzeitigen Test, sowohl mit der Simulationssoftware als auch am realen System, wurden Fehler schnell und frühzeitig erkannt. Insgesamt lässt sich sagen, dass ein zuverlässiges und robustes System geschaffen wurde, das die Ansprüche an ein funktionsfähigen, autonomen Produktionsablauf gerecht wird. Grundsätzlich funktioniert das Ausweichen statischer und dynamischer Hindernisse, sei es Mensch oder Robotino, sowie das Anfahren der Stationen und Ladestationen.

Abschließend kann gesagt werden, dass das Ziel, ein System mit fünf Robotinos zu schaffen, die sich in Ihrem Ablauf selbst organisieren, die über die Ist-Zustände anderer Systemteilnehmer entscheiden, wo sich ihr eigentliches Ziel befindet, ohne zu wissen, wie der Weg und das Ziel der anderen Robotinos sein wird, wurde zufriedenstellend umgesetzt und ist funktionstüchtig.

## 9 Ausblick

Die Validierung hat gezeigt, dass ein zuverlässiges und funktionsfähiges, autonomes System mit der Potentialfeldmethode entstanden ist. Dennoch könnten noch ein paar Verbesserungen an dem Gesamtsystem vorgenommen werden. Eine Voraussetzung des vorliegenden Systems ist, das Hindernisse, wie Stuhlbeine, aus dem Konfigurationsraum entfernt werden, da diese nicht immer von den Infrarot-Sensoren erkannt werden. Die Lösung, die Robotinos kontinuierlich rotieren zu lassen führte in diesem Fall zwar zum Erfolg, beeinträchtigte aber die Qualität des Ausweichmanövers.

Außerdem kommt es zwischenzeitlich noch immer zu Kollisionen, wenn zu viele Robotinos auf einmal aufeinander treffen und der Platz zum Ausweichen zu gering ist. Das Problem ist hier, dass die Robotinos anfangen ein wenig aufzuschwingen, da sie immer wieder in die entgegengesetzte Richtung ausweichen müssen. Lösen könnte man dieses, in dem die Robotinos langsamer werden, sobald mehrere aufeinander treffen. Alternativ könnten weitere Fahrrinnen implementiert werden, so dass es, gerade am Randbereich zwischen Transport- und Fertigungsbereich, nicht zur besagten Situation kommt.

Zudem könnte das Gesamtsystem noch effizienter werden, wenn die Robotinos zwischen zwei gegenüberliegenden Stationen wechseln könnten, ohne den Fertigungsbereich verlassen zu müssen. Das würde dafür sorgen, dass das Verkehrsaufkommen im Transportbereich gemindert wird. Es wäre energetisch und zeitlich effizienter.

# 10 Fazit

Zusammenfassend lässt sich sagen, dass das ausgewählte Konzept, der Potentialfeldmethode, ein sehr sicheres und dynamisches System darstellt. Im Vergleich mit den am Anfang vorgestellten Algorithmen der Bahnplanung besteht dieser Algorithmus aus einem sehr schlankem Grundgerüst. Es wird nicht besonders viel Rechenleistung benötigt. Die Erweiterung des Grundsystems um die einzelnen Kollisionsvermeidungsfeatures ist problemlos möglich.

Die gewählte Entwicklungsmethode, die veränderten oder neu hinzugefügten Features direkt am Robotino zu testen, hat die schnelle Entwicklung des Gesamtsystems begünstigt.

Das Verbundprojekt hat für uns einen sehr großen Lernerfolg mit sich gebracht. Vor allem unsere Software-Skills wurden stark verbessert. Der intensive Umgang mit Matlab/Simulink und damit verbundene Stateflowdiagramme hat unsere Programmierkompetenz stark positiv beeinflusst. Was jedoch unserer Meinung nach völlig im Vordergrund des Verbundprojektes stand, ist die Kommunikation und Zusammenarbeit der verschiedenen Gruppen. Dieses gesamte Projekt war etwas völlig Neues für uns Studenten. Von Anfang an stand das Gelingen im Gesamtprojekt im Vordergrund. Die fünf einzelnen Gruppen haben versucht immer in stetiger Kommunikation zu stehen. An dieser Stelle würden wir gerne noch einmal die wirklich gute Zusammenarbeit mit den Gruppen 3 und 4 hervorheben. Es hat einfach gepasst.

In diesem Projekt sind leider auch negative Seiten einer großen Gruppenarbeit aufgetreten. Die früh aufgestellte Zeitplanung aller Gruppen wurde nicht von jeder Gruppe eingehalten und es kam fast zu einer Situation in der wir nicht hätten vortragen können. Dank intensiver Arbeit von Gruppe 3 und uns kurz vor dem Abschlusstermin konnte der Auftragsdummy noch fertiggestellt werden und eine Präsentation war möglich.

Auch die Betreuung des zuständigen Professors war mehr als zufriedenstellend. Gerade in der Anfangsphase, als es um die Auswahl eines geeigneten Konzeptes ging, aber während der gesamten Umsetzung, stand der Professor unterstützend zur Seite.

Abschließend lässt sich sagen, dass wir drei viel Spaß an der Projektarbeit hatten und auch aus den negativen Erfahrungen positive Aspekte ziehen konnten.

[? ] [? ] [? ] [? ] [? ]