



**FACHHOCHSCHUL-BACHELORSTUDIENGANG
AUTOMATISIERUNGSTECHNIK-INIF**

**Programmierung einer KI für einen autonomen
Roboter**

ALS BACHELORARBEIT EINGEREICHT

zur Erlangung des akademischen Grades

Bachelor of Science in Engineering

von

Bernhard Muckenhumer

06/2010

Betreuung der Bachelorarbeit durch

Prof. (Fh) DI Walter Rokitansky

Ich erkläre ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt, die den benutzten Quellen entnommenen Stellen als solche kenntlich gemacht habe und dass diese Arbeit mit der vom Begutachter beurteilten Arbeit übereinstimmt.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

.....
Bernhard Muckenhumer

Peuerbach, 15.06.2010

KURZFASSUNG

Die Eurobot ist ein internationaler Roboterwettbewerb, der einmal im Jahr in Europa stattfindet. Dafür müssen die Teilnehmer eine vorgeschriebene Aufgabe mit einem autonomen Roboter lösen. Das Robo-Racing-Team¹ der Fh Wels nimmt seit dem Jahr 2007 aktiv an diesem Bewerb teil. Die Eurobot² wird auf der Homepage wie folgt beschrieben:

„Die EUROBOT ist aus dem französischen Nationalen Wettbewerb entstanden. An der EUROBOT können Teams aus allen Ländern der Welt teilnehmen, obwohl der Schwerpunkt und der Austragungsort des Wettbewerbs in Europa liegt. Es nehmen pro Land bis zu drei Teams teil, in den vergangenen Jahren ist die Anzahl der teilnehmenden Mannschaften auf circa 50 gewachsen. EUROBOT findet Ende Mai statt.“

Bei der Eurobot 2010³ sind insgesamt 36 verschiedene Spielfeldkonstellationen möglich, dadurch ist eine dynamische Berechnung einer Fahrtroute von Vorteil. Die Steuersoftware des Roboters soll in der Lage sein durch eine günstige Planung eine Behinderung des Gegners oder eine Kollision weitgehend zu verhindern und dennoch so viele Punkt wie möglich zu erreichen. Nachfolgende Arbeit beschäftigt sich mit der Konzepterstellung und der anschließenden Programmierung der nötigen Treiber, der Ablaufsteuerung und der KI⁴ für den Roboter.

„Wer mit künstlicher Intelligenz arbeitet, muss auch mit natürlicher Dummheit rechnen.“

Klaus Kornwachs (*1947), dt. Hochschullehrer

¹ [Robo-Racing-Team, 2010]

² [Austrobot, 2010]

³ [Regeln Eurobot, 2010]

⁴ [Lämmel, Cleve, S13 2004]

INHALTSVERZEICHNIS

1 MOTIVATION	1
2 ROBOTER- UND SPIELFELDKOMPONENTEN	2
2.1 Roboterkomponenten.....	2
2.2 Spielfeld und Spielelemente	3
3 BESTEHENDE SOFTWARE	4
3.1 Softwarekomponenten	4
3.2 Betriebssystem	4
3.3 Modulkommunikation	5
4 TREIBERERSTELLUNG	6
4.1 Ballerkennung.....	6
4.1.1 Bestimmung der Ballposition.....	6
4.1.2 Nachteile der Ballerkennung	7
4.2 Zylinderentladeschieber.....	7
4.2.1 Nachteile des Zylinderentladeschiebers	7
5 ABLAUFSTEUERUNGSKONZEPT.....	8
5.1 Softwarearchitektur.....	8
5.2 Steuertasks	9
5.2.1 Implementierung eines Tasks	10
5.2.2 Tasksynchronisation.....	10
5.2.3 Taskmodellierung.....	11
5.2.4 Sequenzdiagramm der Orangenernte	13
6 ERSTELLUNG DER KI.....	14
6.1 Konzept.....	14
6.1.1 Einflussfaktoren.....	15
6.1.2 Prioritätenvorgabe	15
6.1.3 Fahrtroutenberechnung der KI.....	16
6.1.4 Korrekturfahrten und Koordinatenoffsets	16
6.2 Modellierung der KI	17
6.3 Einbindung der KI in die Ablaufsteuerung.....	19

7 ZUSAMMENFASSUNG UND AUSBLICK	20
8 LITERATUR.....	21
8.1 Bücher.....	21
8.2 Internetquellen	21
8.3 Relevante Bachelorarbeiten	21
8 ANHANG	22
8.1 C-Code der KI.....	22

ABBILDUNGSVERZEICHNIS

Abb. 1: Robotergesamtansicht 1	2
Abb. 2: Robotergesamtansicht 2	2
Abb. 3: Spielfeld	3
Abb. 4: Zur Verfügung stehende Softwarekomponenten.....	4
Abb. 5: Übersicht des modularen Systems V2.0.....	5
Abb. 6: Schematische Darstellung der Ballerkennung.....	6
Abb. 7: Softwarearchitektur	8
Abb. 8: Erstellte Tasks	9
Abb. 9: Bereitgestellte Tasks	9
Abb. 10: Initialisierungsteil eines Tasks	10
Abb. 11: Funktionsteil eines Tasks	10
Abb. 12: Tasksynchronisation im Hauptsteuertask.....	11
Abb. 13: Transitionsbedingungen für den Collect-Tomatoes-Task.....	12
Abb. 14: Zustandsautomat für die Tomatenaufnahme	12
Abb. 15: Sequenzdiagramm für die Orangenernte.....	13
Abb. 16: Darstellung der Rastereinteilung des Spielfeldes.....	14
Abb. 17: Definition der Elementarten.....	15
Abb. 18: Berechnete Fahrtroute der KI bei gegebener Konstellation	16
Abb. 19: Zustandsautomat der KI	17
Abb. 20: Darstellung der Arbeitsweise der KI.....	18
Abb. 21: Einbindung der KI in den Hauptsteuertask	19

1 MOTIVATION

Das Robo-Racing-Team hat bereits zahlreiche autonome Roboter für verschiedene Bewerbe entwickelt und diese erfolgreich eingesetzt. Aufgrund der Aufgabenstellung für die Eurobot 2010 ist es vorteilhaft, wenn die Steuersoftware eigenständig eine Fahrtroute planen kann. Diese Fahrtroutenplanung ist von verschiedenen Umständen, wie der aktuellen Position des Gegners, der Anzahl der bereits geladenen Elemente und der noch vorhandenen Spielzeit abhängig. Aufgrund des für diesen Bewerb entwickelten Kamerasystems⁵ zur Erkennung der Spielfeldkonstellation, kann eine KI dynamisch den optimalen Weg berechnen. Es gibt dadurch nur eine Strategie, die selbstständig je nach Situation die nächste Aktion beziehungsweise den nächsten Anfahrtpunkt bestimmt.

⁵ [Muckenhumer, 2010]

2 ROBOTER- UND SPIELFELDKOMPONENTEN

2.1 Roboterkomponenten⁶

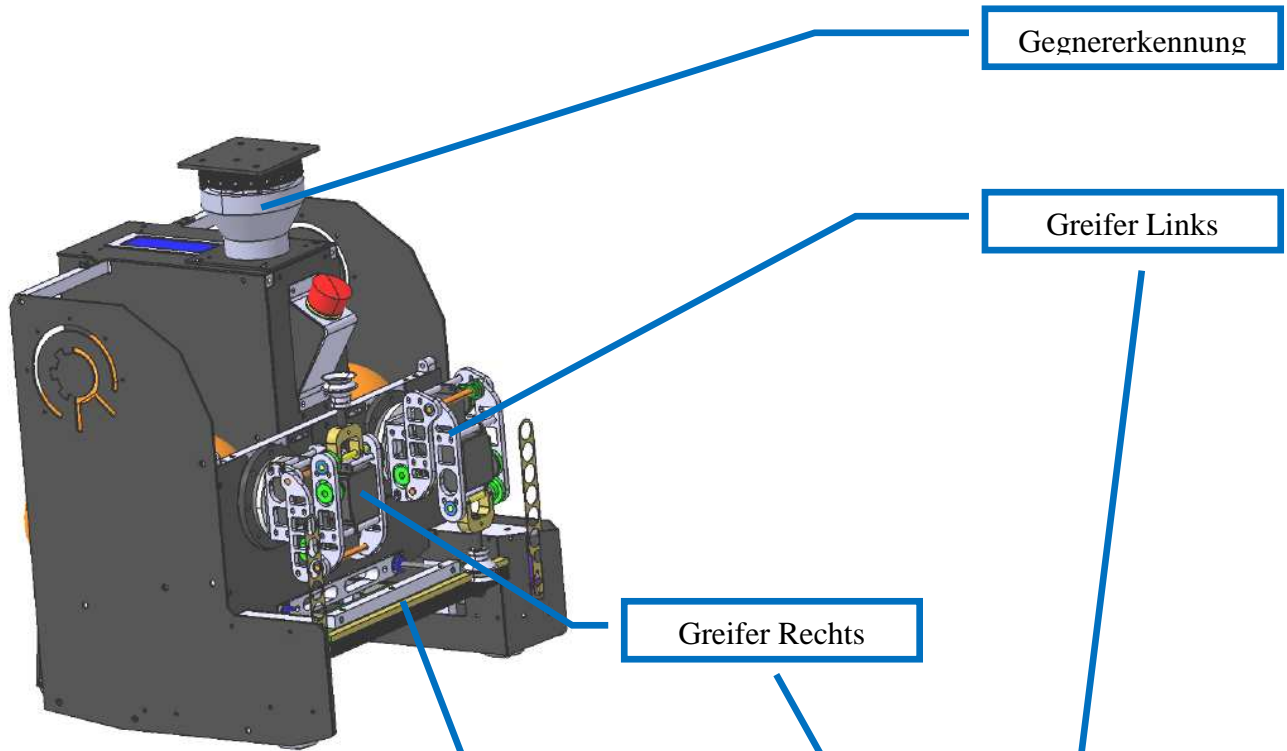


Abb. 1: Robotergesamtansicht 1

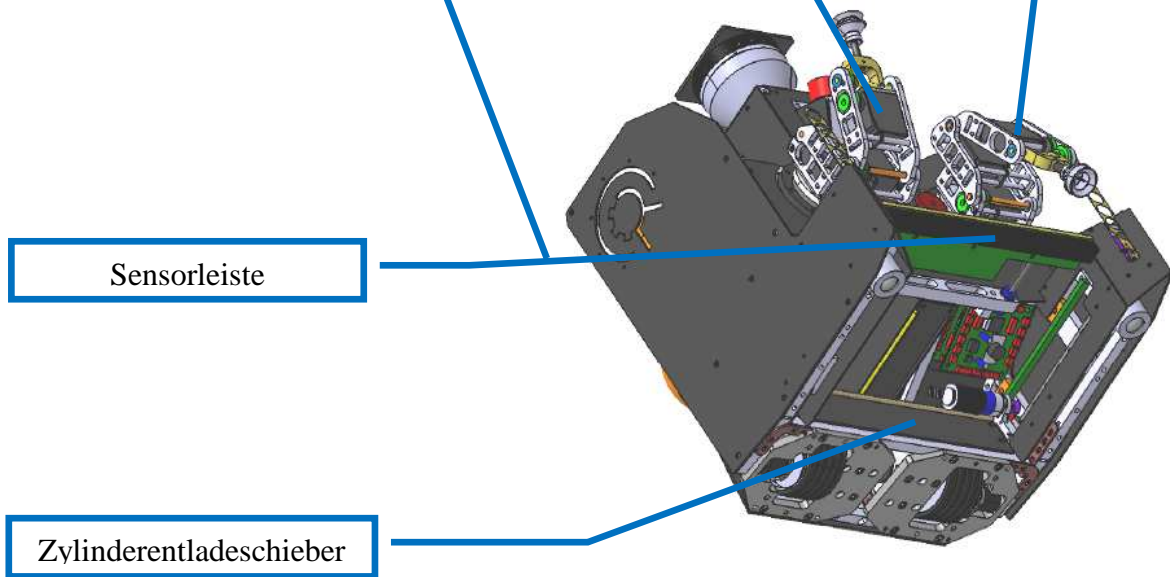


Abb. 2: Robotergesamtansicht 2

⁶ [Pautzenberger, 2010]

2.2 Spielfeld und Spielelemente⁷

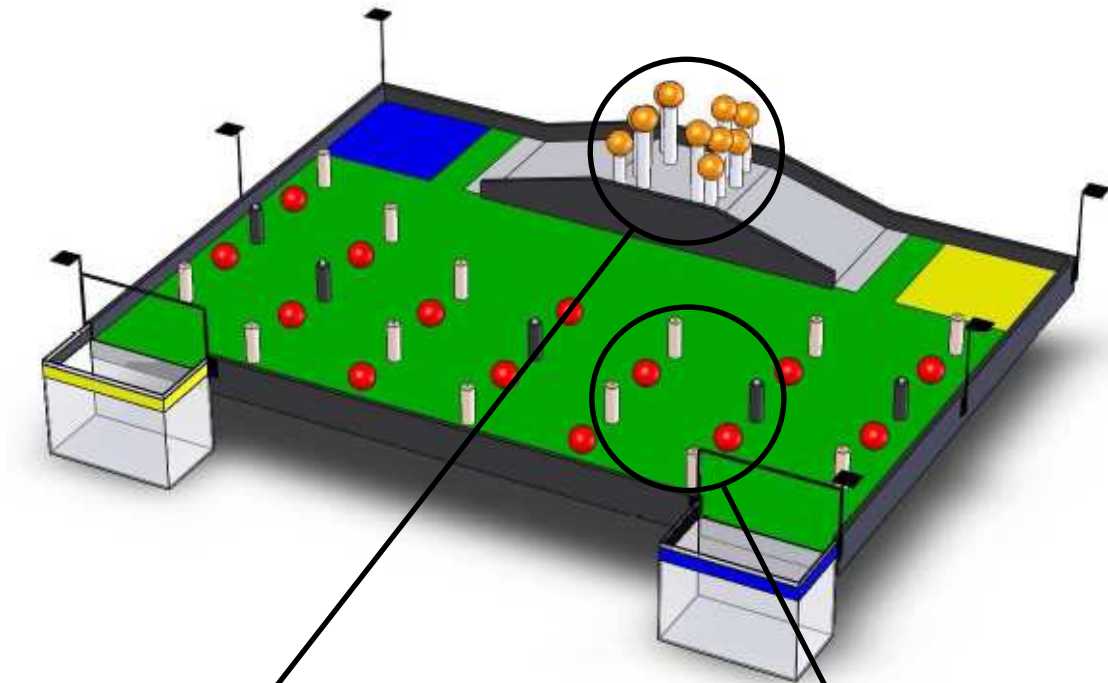
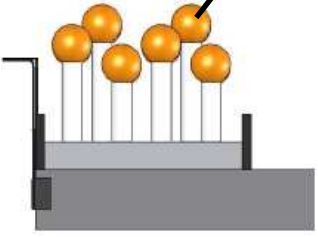
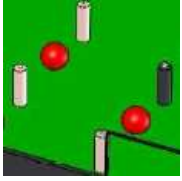


Abb. 3: Spielfeld

	
<p>Auf jeder Rampe sind sechs Orangen auf Kunststoffzylindern platziert.</p> <p>1 Orange = 300 Punkte</p>	<p>Auf dem Spielfeld sind „ears of corn“, „fake-corns“ und „tomatoes“ verteilt.</p> <p>1 ear of corn = 250 Punkte</p> <p>1 tomatoe = 150 Punkte</p> <p>Die schwarzen „fakecorns“ können nicht eingesammelt werden und sind fest verschraubt.</p>

⁷ [Regeln Eurobot, 2010]

3 BESTEHENDE SOFTWARE

Der Roboter wird mit der modularen Systemelektronik V2.0⁸ ausgestattet, dessen Treiberprogramme bereits vorhanden sind. Für die Eurobot 2010 werden jedoch eine Ballerkennung sowie ein Zylinderentladeschieber vorgesehen, für welche die Treiberprogramme erstellt werden müssen.

3.1 Softwarekomponenten

Folgende Softwarekomponenten werden zur Verfügung gestellt⁹ (Abb. 4):

Name der Komponente	Anwendung für den Roboter
adc.c	Einlesen der analogen Drucksensoren für die Greifarmsauger
uart.c	Serielle Kommunikation zu den Beacons ¹⁰ und zum debuggen
spi.c	Kommunikationsprotokoll zwischen den elektronischen Modulen des Roboters
rtlan.c	Datenaustausch zwischen den elektronischen Modulen des Roboters
genTraj.c	Trajektorien-Berechnung für den Antriebsregler
drive.c	Anfahren eines Punktes nach Vorgabe der KI
map.c ¹¹	Mapping für die Gegnerposition und die eigene Position

Abb. 4: Zur Verfügung stehende Softwarekomponenten

3.2 Betriebssystem

Das Steuerprogramm für den Roboter wird auf Basis eines kooperativen Multitasking-Betriebssystems¹² entwickelt und kann dadurch in verschiedene Steuertasks aufgliedert werden. Dies erleichtert die Entwicklung und Übersichtlichkeit des Steuerprogramms und stellt eine komfortable Möglichkeit dar, einzelne Tasks in künftigen Projekten wiederzuverwenden.

⁸ [Zauner, 2009]

⁹ [Zauner, 2009]

¹⁰ [Regeln Eurobot, 2010]

¹¹ [Weinberger, 2010]

¹² [Zauner, 2009]

3.3 Modulkommunikation

In Abb. 5 ist die Datenverbindung zwischen den Modulen via SPI¹³ ersichtlich. Das kooperative Multitasking-Betriebssystem ist jeweils auf dem Main-Board und dem LCD-Board implementiert.

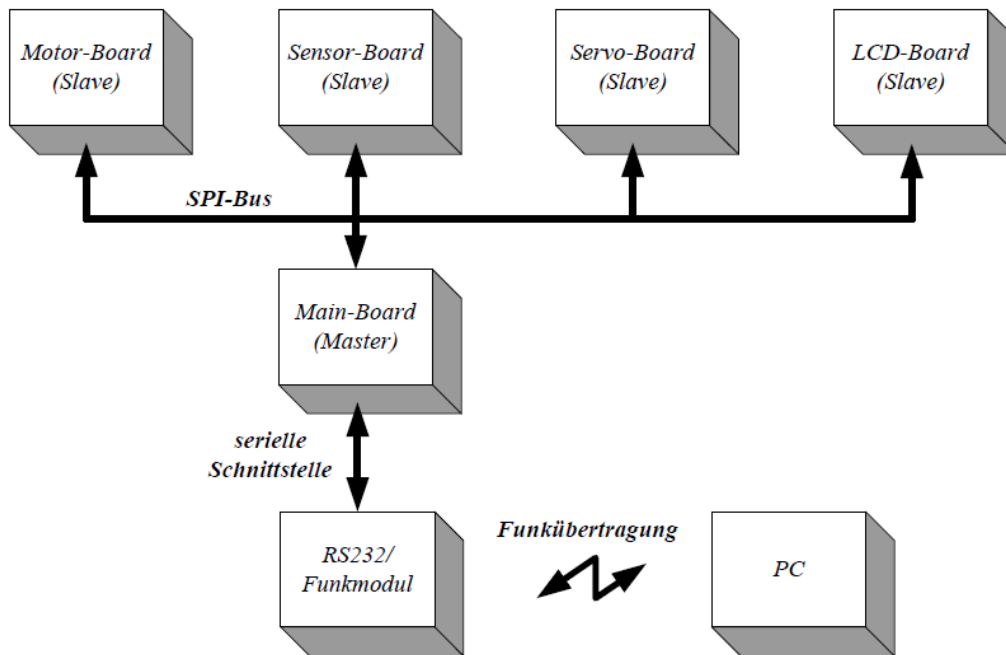


Abb. 5: Übersicht des modularen Systems V2.0

¹³ [G.Schmitt, S376 – S388, 2006]

4 TREIBERERSTELLUNG

Wie in Punkt 3 *Bestehende Software* erwähnt, müssen für die Eurobot 2010 Treiber für die Ballerkennung und den Zylinderentladeschieber erstellt werden. Nachfolgend wird die Erstellung kurz beschrieben.

4.1 Ballerkennung

Für die Ballerkennung ist an der Front des Roboters eine Sensorleiste (siehe Punkt 2.1 *Roboterkomponenten*) angebracht, die mit fünfzehn digitalen Infrarotsensoren des Typs GP2Y0G340K ausgestattet ist. Durch die Tatsache, dass bei einem Ball (Tomate), der vor der Sensorleiste liegt, fünf oder sechs Sensoren ansprechen, kann eine Ballerkennung entwickelt werden, die diese Eigenschaften auswertet.

4.1.1 Bestimmung der Ballposition

Mit Hilfe der Anzahl und der Nummern der angesprochenen Infrarotsensoren kann eine Entfernung in y-Richtung vom Nullpunkt bestimmt werden (siehe Abb. 6). Die x-Richtung kann nur angenommen werden und wird mittels Offset angegeben. Die errechneten Koordinaten werden dem jeweiligen Greifertask übergeben.

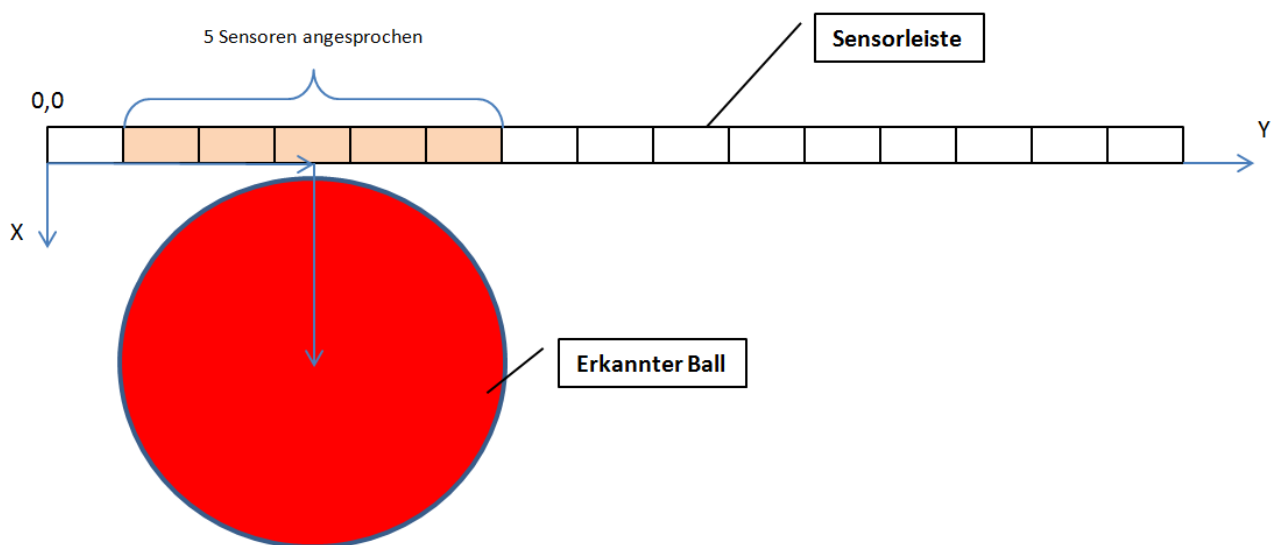


Abb. 6: Schematische Darstellung der Ballerkennung

Die Breite eines Infrarotsensors (15,2mm) wird mit der Nummer des ersten angesprochenen Sensors multipliziert. Anschließend wird zum Ergebnis noch die Entfernung zum Mittelpunkt addiert. Um eine Fließkommaberechnung zu vermeiden wurde die Berechnung wie folgt implementiert.

```
uiDistanceBall=(unsigned int)(((76*uiStartCoordinate)/5)+((76*uiAnzahlSens)/10))
```

Die so berechnete Distanz in y-Richtung erreicht eine Genauigkeit von $\pm 2\text{mm}$ Abweichung von der tatsächlichen Distanz.

Erläuterung zur Berechnung:

uiStartCoordinate Nummer des ersten angesprochenen Sensors vom Nullpunkt

uiAnzahlSens Anzahl der zusammenhängenden angesprochenen Sensoren

$76/5 = 15,2$ Breite eines Infrarotsensors

$(76 * \text{uiStartCoordinate})/5 =$ Entfernung vom Nullpunkt bis zum ersten angesprochenen Sensor

$(76 * \text{uiAnzahlSens})/10 =$ Anzahl der angesprochenen Sensoren dividiert durch die Breite eines Sensors dividiert durch 2, um den Ballmittelpunkt zu erhalten

4.1.2 Nachteile der Ballerkennung

Die Infrarotsensoren besitzen eine Reichweite von ca. 15cm, was bei Fahrtgeschwindigkeiten $>0.25\text{m/s}$ des Roboters zu einer verspäteten Erkennung der Bälle führt. Um den Ball mit dem Greifer aufnehmen zu können, muss der Roboter zum Stillstand kommen, bevor dieser den Ball berührt. Andernfalls wird der Ball vom Roboter weggestoßen und kann nicht aufgenommen werden.

4.2 Zylinderentladeschieber

Der Zylinderentladeschieber wird mittels Linearantrieb mit einem DC Motor der Firma Faulhaber angetrieben. Bei der Systeminitialisierung wird dieser in die hintere Endlage gesteuert. Im Zuge der Ausladeautomatik des Roboters vor dem Ziel, wird der Zylinderentladeschieber in die vordere Endlage gesteuert und anschließend wieder in die Hintere. Die Steuerkommandos werden dafür vom Main-Board an das Motor-Board des Zylinderentladeschiebers gesendet. Um Störungen zu erkennen, wird der Schieber bei der Ansteuerung Zeitüberwacht und führt die Tätigkeit erneut aus, wenn nach einer Zeit von 3s kein Endlagenschalter eingelesen wird.

4.2.1 Nachteile des Zylinderentladeschiebers

Die vordere Endlage des Zylinderentladeschiebers ist zu weit im Inneren des Roboters, was zur Folge haben kann, dass nicht alle Zylinder ins Ziel rollen.

5 ABLAUFSTEUERUNGSKONZEPT

Die Ablaufsteuerung des Roboterprogramms wird von einem Hauptsteuertask ausgeführt, der je nach ausgewählter Strategie in Verbindung mit der KI die Fahrroutenplanung und die Aktivierung der anderen Tasks übernimmt. Die Information der ausgewählten Spielfarbe wird über die Einstellung am LCD jedem Task zugänglich gemacht und kann somit Taskunabhängig ausgewertet werden.

5.1 Softwarearchitektur

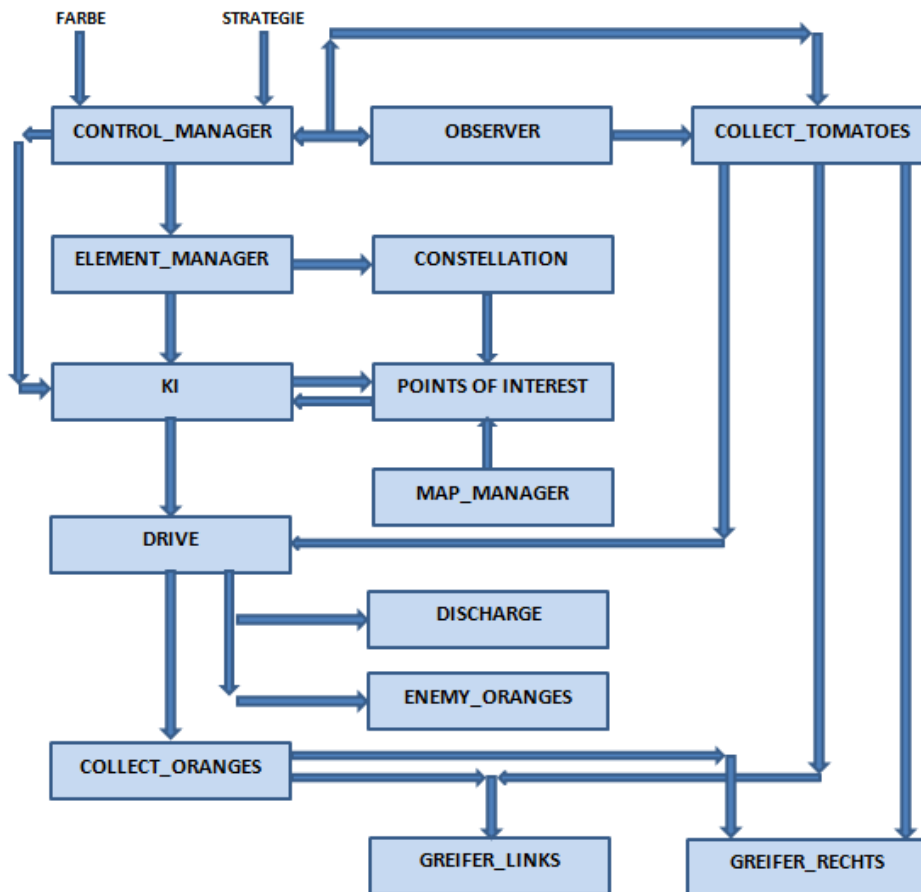


Abb. 7: Softwarearchitektur

In Abb. 7 werden die Beziehungen zwischen den einzelnen Steuertasks gezeigt. Eine Ausnahme bildet der Block „POINTS OF INTEREST“, welcher einen globalen Speicher für die Position und Art der Elemente beziehungsweise Punkte auf dem Spielfeld darstellt. Unter Punkt 4.2 *Steuertasks* sind die Tasks und deren Aufgaben angeführt.

5.2 Steertasks

Steuertasks, die im Zuge dieser Bachelorarbeit erstellt worden sind (Abb. 8):

Name des Tasks	Funktion im Roboter
DEBUG	Für das Testen der Aktorik, Sensorik und der Steertasks
CONTROL_MANAGER	Hauptsteertask, der je nach Strategie die Steuerinformationen vergibt und die Tasks aktiviert
ELEMENT_MANAGER	Elementverwaltungstask für die Spielelemente auf dem Tisch
OBSERVER	Ballerkerkennung und Koordinatenberechnung
COLLECT_ORANGES	Steuertask für die Orangenernte auf den Rampen
COLLECT_TOMATOES	Steuertask für das Aufnehmen der Tomaten
SLIDER	Steuertask für das Ansteuern und Überwachen des Zylinderentlade-schiebers
CONSTELLATION	Ermittlung der aktuellen Konstellation am Spielfeld durch Anforderung der Kameradatenpakete ¹⁴ von den Beacons
ENEMY_ORANGES	Ermittlung der noch vorhandenen Orangen auf der Gegnerrampe durch Anforderung des Kameradatenpaketes
DISCHARGE	Ausladeautomatik für das Entleeren der eingesammelten Spielelemente

Abb. 8: Erstellte Tasks

Steuertasks, die im Zuge anderer Bachelorarbeiten erstellt worden sind (Abb. 9):

Name des Tasks	Funktion im Roboter
GREIFER_LINKS	Steuertask für den Greifer links
GREIFER_RECHTS ¹⁵	Steuertask für den Greifer rechts
DRIVE	Antriebssteuertask für das Punktanfahren
MAP_MANAGER ¹⁶	Erstellen eines Mappings der Fahrtroute des Gegners und des eigenen Roboters, dies wird unter anderem von der KI ausgewertet

Abb. 9: Bereitgestellte Tasks

¹⁴ [Muckenhumer, 2010]

¹⁵ [Rathgeber, 2010]

¹⁶ [Weinberger, 2010]

5.2.1 Implementierung eines Tasks

Die Softwareabbildung jedes Tasks ist grundsätzlich gleich. Ein Task besteht aus einem Initialisierungsteil, welcher die Art festlegt und einer Steuerfunktion, die bei der Initialisierung als Funktionszeiger übergeben wird. In Abb. 10 und 11 ist der grundsätzliche C-Code eines Tasks dargestellt.

```
void InitTask(void)
{
    // zyklischer Task - Zykluszeit: 500 ms
    SET_CYCLE(Task, 500);
    SET_TASK(Task, CYCLE);
    SET_TASK_HANDLE(Task, Taskfunction);
}
```

Abb. 10: Initialisierungsteil eines Tasks

```
unsigned char Taskfunction(void)
{
    //Zykluszeit setzen
    SET_CYCLE(Task, 500);

    //Steuercode

    return(CYCLE);
}
```

Abb. 11: Funktionsteil eines Tasks

5.2.2 Tasksynchronisation

In vielen Fällen ist es notwendig, dass die verschiedenen Steuertasks Informationen untereinander austauschen können. Dafür ist für jeden Task durch das Betriebssystem eine Mailbox vorgesehen, die einen Speicher von 10 Byte umfasst. Diese Mailboxen sind global schreib- und lesbar. Folgendes Beispiel (Abb. 12) zeigt wie der Hauptsteuertask darauf wartet, bis die aktuelle Konstellation des Spielfeldes ermittelt worden ist.

```

//*****
//State 200: ElementManager aktivieren //
//*****
else if(ucStateControlManager == 200)
{
    InitElementManager();
    ucStateControlManager = 201;
}
//*****
//State 201: KI initialisieren //
//*****
else if(ucStateControlManager == 201)
{
    //Elemente wurden eingelesen und POI Struktur wurde initialisiert
    if(READ_FROM_MAILBOX(ELEMENT_MANAGER_TASKNBR, STATUS) == READY)
    {
        ucStateControlManager = 202;
        Init_KI();
        InitMapManager();
    }
}

```

Abb. 12: Tasksynchronisation im Hauptsteuertask

Für die Abfrage in welchem Satus sich ein Steuertask befindet wird ein STATUS-Byte angelegt, welches anschließend im jeweiligen Task beschrieben wird und im Hauptsteuertask gelesen und ausgewertet wird. Auf diese Art kann ein Task einem anderen verschiedenste Informationen bereitstellen bevor er gestartet wird oder auch während dieser aktiv ist.

5.2.3 Taskmodellierung

Alle Steuertasks werden ausnahmslos als Zustandsautomaten modelliert und abgebildet. Im Folgenden Beispiel wird die grundsätzliche Modellierung anhand des „Collect-Tomatoes-Task“ erläutert.

1. Zustände definieren:

Zustand	Definition
1	Observertask auf erkannte Tomaten abfragen
2	Wenn eine Bewegung aktiv ist, diese unterbrechen und Zwischenspeichern
3	Abfragen ob eine oder zwei Tomaten erkannt wurden und entsprechende Greifer starten
4	Wenn die Tomate aufgenommen wurde, abfragen ob weitere Tomaten erkannt wurden

Die Taskfunktion besteht somit aus vier Zuständen. Durch die Transitionsbedingung wird von einem Zustand in einen anderen gewechselt. Jedem Zustand ist eine Tätigkeit zugeordnet.

2. Transitionsbedingungen definieren:

Von Zustand	In Zustand	Bedingung
1	2	Tomate wurde erkannt
2	3	Bewegung wurde abgebrochen
3	4	Tomate wurde aufgenommen
4	1	Keine weitere(n) Tomate(n) erkannt
4	3	Weitere Tomate(n) wurde erkannt

Abb. 13: Transitionsbedingungen für den Collect-Tomatoes-Task

3. Grafische Darstellung des Zustandsautomaten:

Abb. 14 zeigt den Zustandsautomaten für die Tomatenaufnahme.

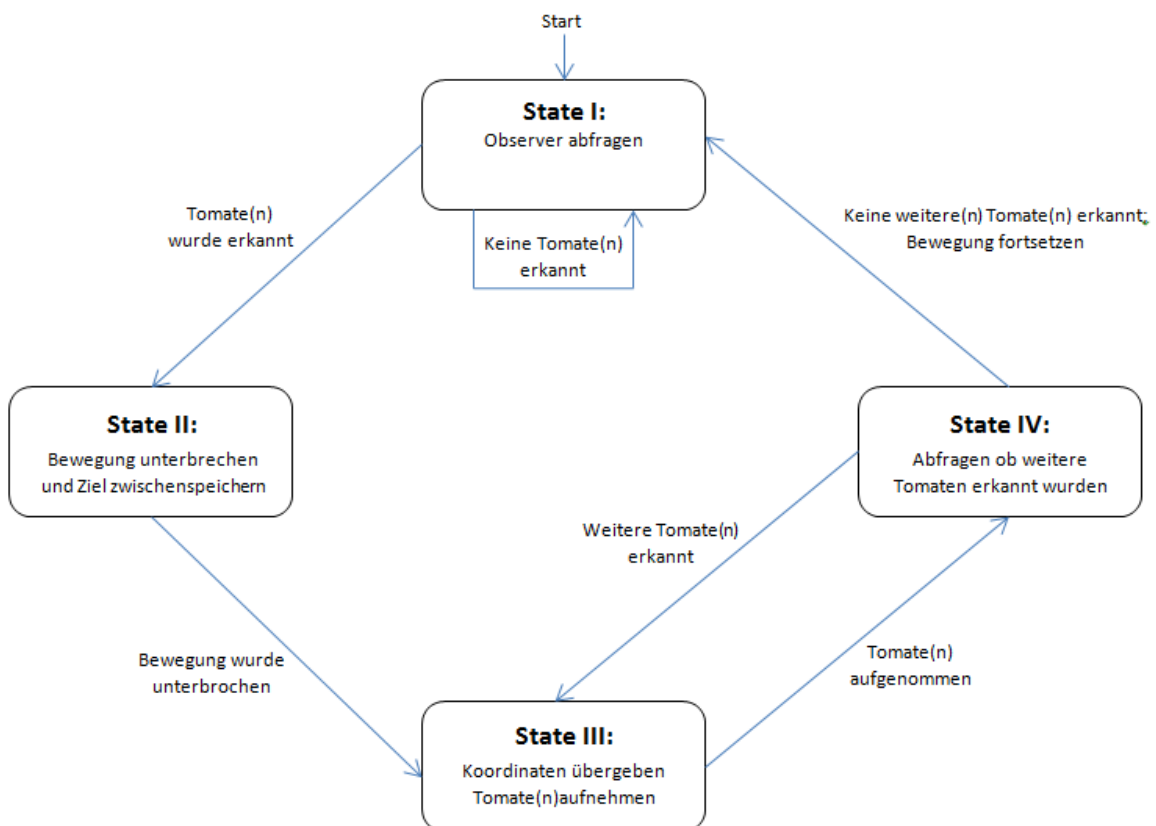


Abb. 14: Zustandsautomat für die Tomatenaufnahme

5.2.4 Sequenzdiagramm der Orangenernte

Nachfolgendes Sequenzdiagramm (Abb. 15) zeigt die Abfolge der Steuertasks für die Orangenernte (Spielfarbe Gelb). Es wird gezeigt, wie das Zusammenwirken der Tasks die Gesamtfunktion der Orangenernte realisiert.

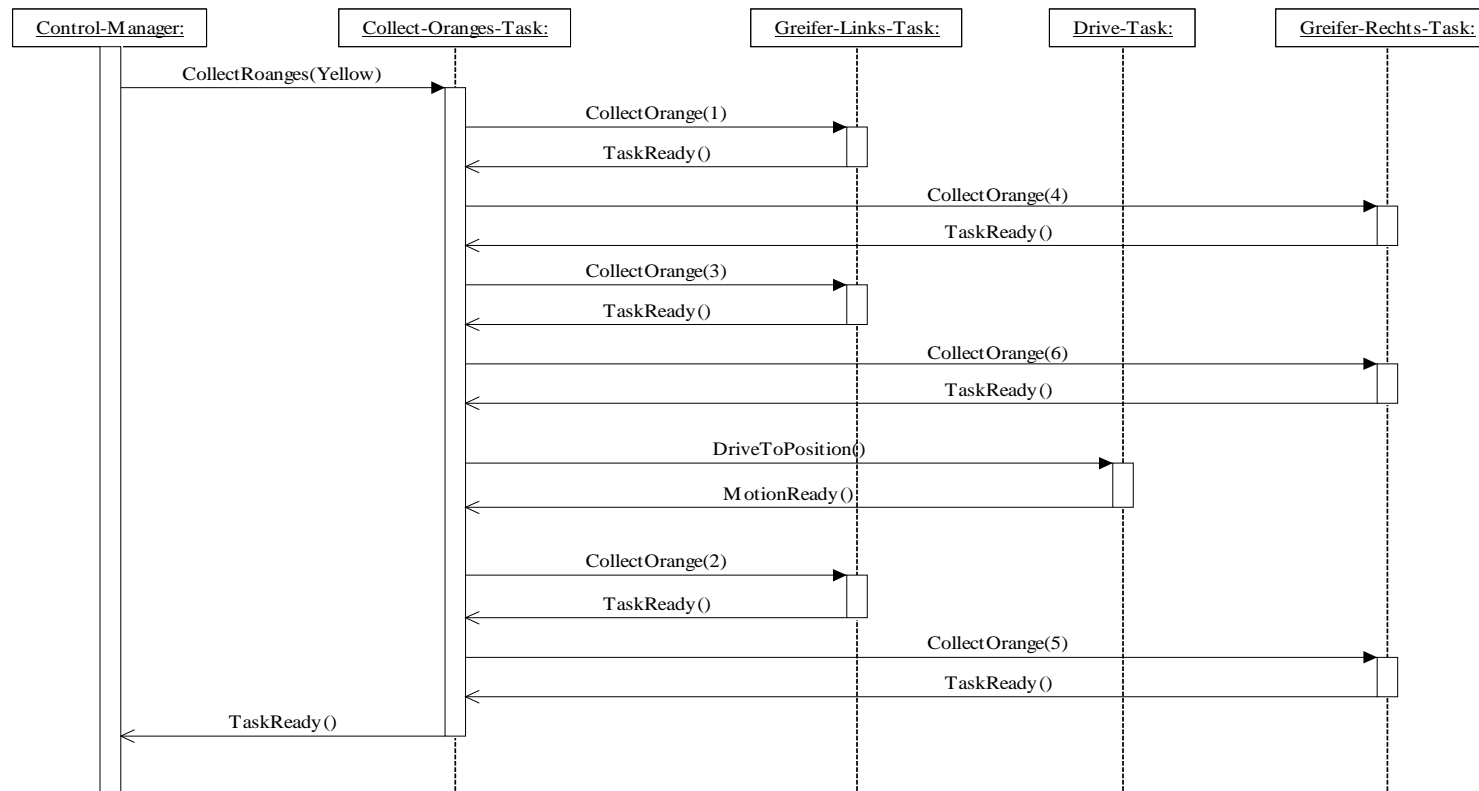


Abb. 15: Sequenzdiagramm für die Orangenernte

6 ERSTELLUNG DER KI

Das Herzstück der Fahrtroutenplanung bildet die KI, welche eigenständig unter Rücksichtnahme verschiedener, situationsbedingter Einflussfaktoren einen günstigen Weg zur Entladezone ermittelt. In diesem Kapitel wird das Vorgehen vom Konzept bis zur Implementierung der KI vorgestellt.

6.1 Konzept

Für die Berechnung einer günstigen Fahrtroute wird das Spielfeld in ein Raster unterteilt, dessen Einzelfelder einem POI (Point of interest) entsprechen. Jedes Einzelfeld wird mit den Positionskoordinaten (x, y) , einer Elementart und einem Steuerbefehl belegt. Dieser Steuerbefehl bestimmt den aufzurufenden Task nach dem Anfahren eines Punktes. So kann zum Beispiel realisiert werden, dass automatisch nach dem Anfahren des Punktes vor dem Ziel die Ausladeautomatik gestartet wird. In Abb. 16 ist das Raster ersichtlich.

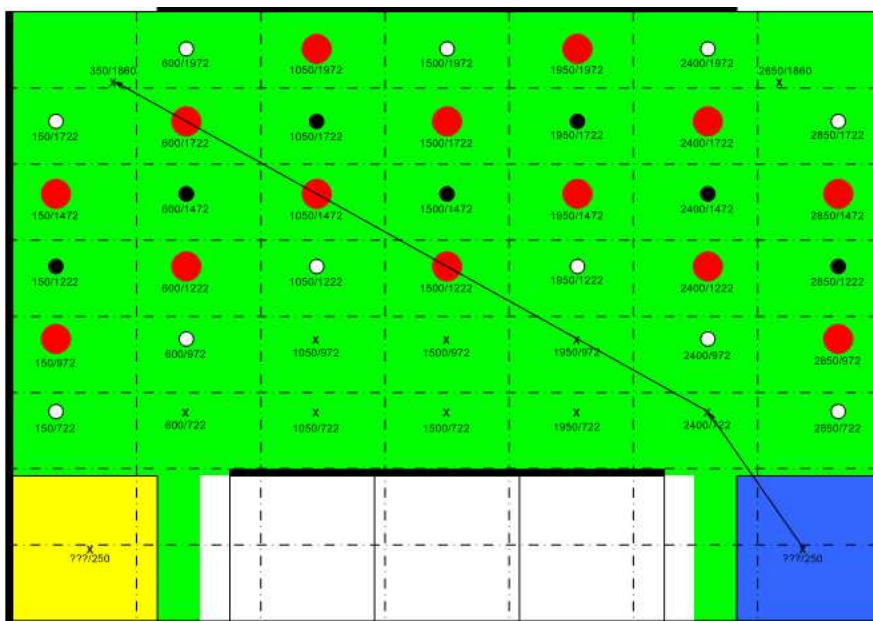


Abb. 16: Darstellung der Rastereinteilung des Spielfeldes

Die eingezeichnete Fahrtroute wird für die Homolegation verwendet, wenn die Spielfarbe Blau ist. Für Gelb gilt die gespiegelte Route. Bei der Homolegation muss bewiesen werden, dass der Roboter aus eigener Kraft mindestens einen Punkt machen kann und eine funktionierende Gegnererkennung besitzt. Andernfalls wird er nicht zum Bewerb zugelassen.

6.1.1 Einflussfaktoren

Für die Erstellung der KI müssen im Vorfeld die Einflussfaktoren bestimmt werden, von denen die Steuerung abhängig ist:

- Position des gegnerischen Roboters
- Position des eigenen Roboters
- Noch vorhandene Spielzeit
- Anzahl der geladenen Orangen/Tomaten/Zylinder
- Mapping (Wo war der Gegner bereits und wo war der eigene Roboter bereits)

6.1.2 Prioritätenvorgabe

Aufgrund der vorher genannten Einflussfaktoren werden eine priorisierte Fahrtrichtung und eine priorisierte, zu sammelnde Elementart generiert. Durch die Definition der Elementarten mittels Zahlenwert kann das priorisierte Element ermittelt werden (siehe Abb. 17).

```
#define CORNEAR          0
#define TOMATOE         1
#define NAVPOINT        2
#define ENEMY_ROBOT     3
#define OWN_ROBOT       4
#define FAKECORN        5
```

Abb. 17: Definition der Elementarten

Aus Abb. 17 ist zusätzlich ersichtlich, dass auch Navigationspunkte, die an sich kein Spielelement sondern nur einen Anfahrtspunkt beinhalten, in diese Prioritätsvorgabe einfließen. Wurde ein Spielelement bereits vom Gegner oder vom eigenen Roboter eingesammelt, wird der Zahlenwert in der POI-Struktur angepasst und die KI erkennt somit, dass die Priorität niederwertiger eingestuft wurde. Unterschreitet die noch vorhandene Spielzeit einen gewissen Wert (z.B.: 25s) ist die primäre Aufgabe der KI nicht mehr die maximalen Punkte zu sammeln, sondern auf schnellstem Wege zum Ziel zu gelangen. In diesem Fall werden Navigationspunkte gegenüber Elementen bevorzugt, wenn sie der primären Fahrtrichtung entsprechen.

6.2 Modellierung der KI

Auch die KI ist als Zustandsautomat ausgeführt und wird wie folgt aufgebaut.

1. Priorisierte Fahrtrichtung und priorisiertes Element ermitteln
2. Mögliche Anfahrtpunkte vom Roboterstandpunkt und vom Gegnerstandpunkt ausgehend ermitteln
3. Den aufgrund der Prioritätenvorgabe günstigsten Anfahrtpunkt auswählen
4. Punkte wenn möglich überschleifen, zum Beispiel wenn diese in einer Linie liegen
5. Koordinatenoffsets ermitteln oder wenn nötig eine Korrekturfahrt einleiten
6. Den ersten Anfahrtpunkt vom erstellten Array anfahren und wieder mit Zustand 1 beginnen

In Abb. 19 ist der Zustandsautomat abgebildet.

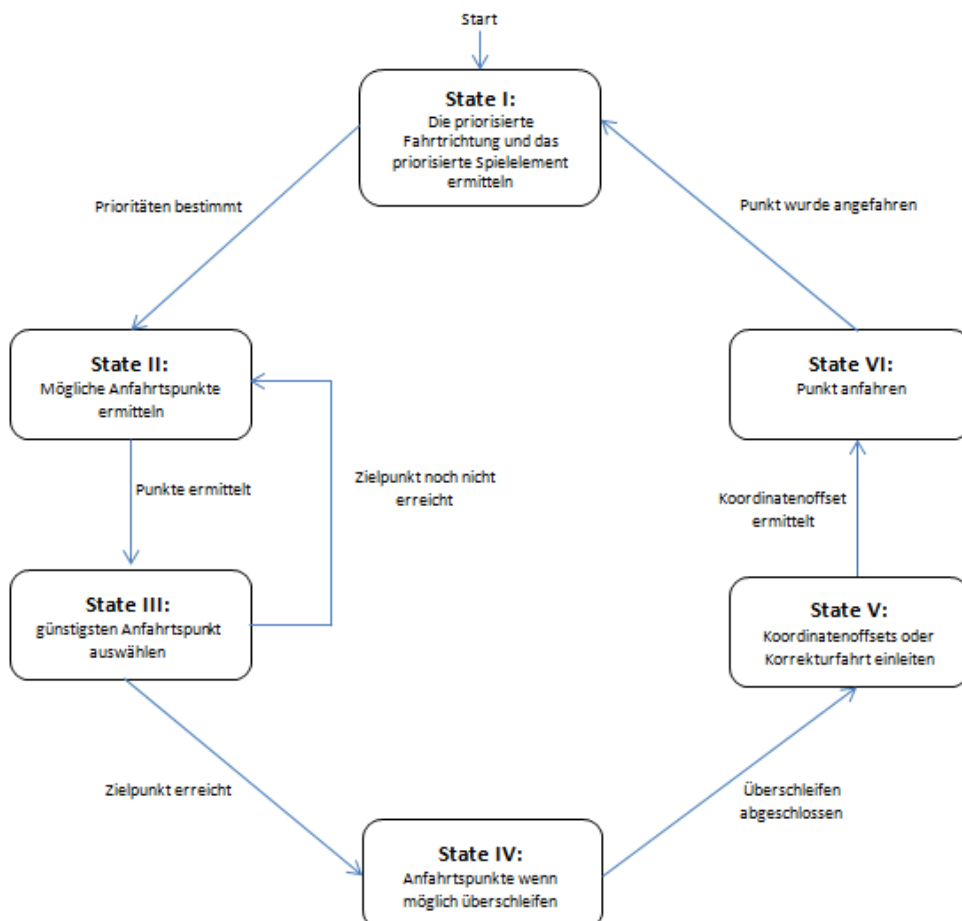


Abb. 19: Zustandsautomat der KI

In der folgenden Abbildung (Abb. 20) soll die Ermittlung der möglichen Anfahrtspunkte verdeutlicht werden. Der markierte Teil des Spielfeldes um den Roboter ist jener Teil, den die KI für die Berechnung des nächsten Anfahrtspunktes in Betracht zieht. Das punktierte Rechteck ist der POI, der als nächster Anfahrtspunkt ausgewählt wurde. Befindet sich der Gegner innerhalb des größeren markierten Rechteckes, so trägt das Mapping diese Position ins POI ein und die KI erkennt diese Position nicht mehr als möglichen Anfahrtspunkt.

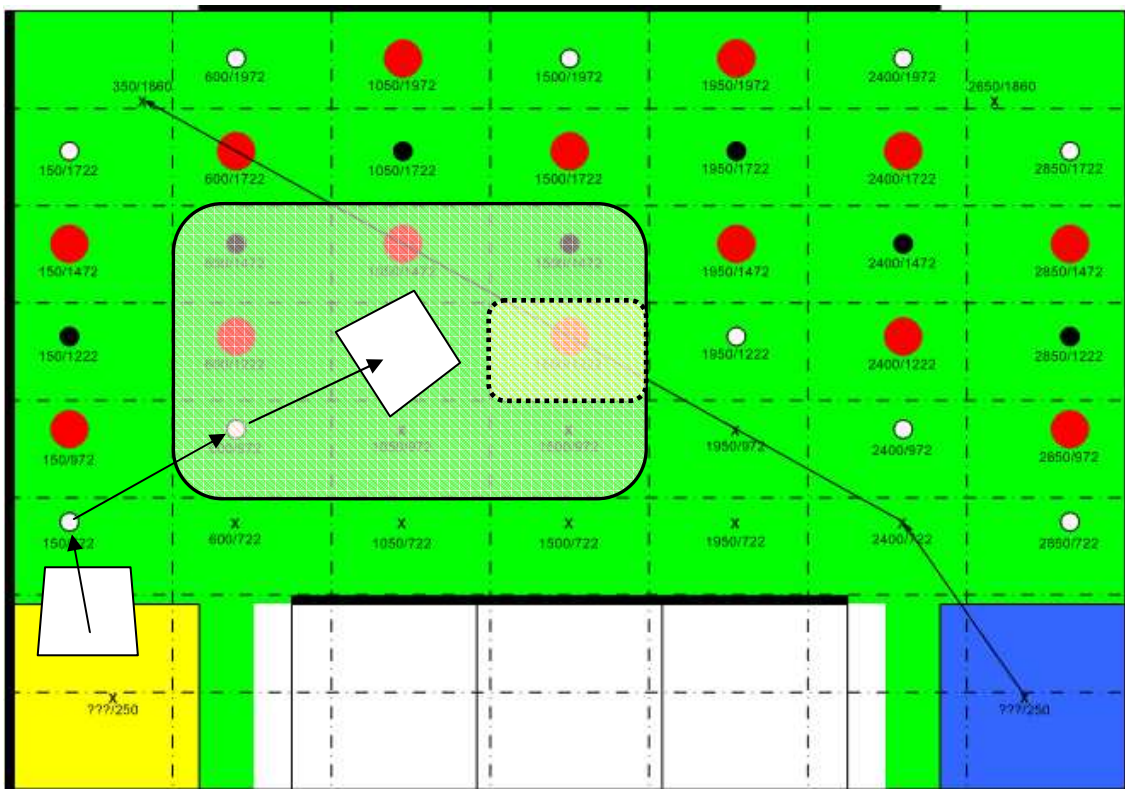


Abb. 20: Darstellung der Arbeitsweise der KI

Der Vorteil bei dieser Modellierung ist das modulare Prinzip eines Zustandsautomaten. Es kann problemlos ein Zustand angepasst oder hinzugefügt werden. Auch die Transitionsbedingungen können sehr einfach angepasst werden.

6.3 Einbindung der KI in die Ablaufsteuerung

Die KI wird im Hauptsteuertask als Funktion „SetNextPointOfInterest()“ eingebunden. Sie bleibt ein in sich geschlossenes Teilsystem der Ablaufsteuerung und wird vom Hauptsteuertask als Unterzustandsautomat aufgerufen. Im Hauptsteuertask gibt es für die KI einen zentralen Zustand, der die Positionsvorgaben der KI ausführt (siehe Abb. 21).

```

//*****
//State 203: Befehlsvorgabe der KI abfragen //
//*****
else if(ucStateControlManager == 203)
{
    //KI berechnet den nächsten Anfahrtpunkt
    SetNextPointOfInterest();
}
//*****
//State 204: Befehlsausführung //
//*****
else if(ucStateControlManager == 204)
{
    //Der nächste zu startende Task nach dem Punktanfahren wird ausgelesen
    ucInstructionTask = NextPoint.ucInstruction;
    ucErrorCaller = CONTROL_MANAGER_TASKNBR;

    DriveToPosition(((float)NextPoint.uiXposition)/1000,
                    ((float)NextPoint.uiYposition)/1000,
                    fSpeed*fDirection, ON, ucInstructionTask);

    ucStateControlManager = 203;

    return(DISABLE);
}
//*****

```

Abb. 21: Einbindung der KI in den Hauptsteuertask

Die KI kann somit einfach in künftige Projekte portiert werden, da sie nicht verteilt implementiert ist.

7 ZUSAMMENFASSUNG UND AUSBLICK

Die Programmierung einer KI für einen autonomen Roboter macht diesen sehr flexibel, da die KI für verschiedenste Aufgaben angepasst und erweitert werden kann. Durch die Aufteilung der Steueraufgaben in Tasks wird die Übersichtlichkeit und Wiederverwendbarkeit sehr einfach und effizient. Der Nachteil der KI, die für die Eurobot 2010 erstellt wurde ist, dass diese sehr viele Informationen über die aktuelle Spielfeldkonstellation benötigt, was die aufwendige Maßnahme der Kamera-Beacons zur Folge hatte. Durch die Bauleranzen der Bewerbungstische müssen vor jedem Match die Kameras justiert werden, was in den vorgegebenen drei Minuten nicht immer bewerkstelligt werden kann. Die KI konnte aufgrund von massiven Problemen mit dem Antrieb nicht ausreichend getestet und deshalb auch nicht fertiggestellt werden.

Im Zuge des Berufspraktikums an der Fh Wels hat sich herausgestellt, dass das modulare System V2.0 nicht für die endgültige Steuerelektronik eines Roboters für die Eurobot geeignet ist. Aufgrund der erstmals eingesetzten digitalen Servomotoren für die Greifarme entstehen hohe Einschaltströme, was zu einem Versagen der SPI Verbindung zwischen den Modulen führen kann. Weiters ist der Verdrahtungsaufwand der durch ein derartiges modulares System entsteht sehr hoch. Die Eurobot 2010 hat gezeigt, dass Teams, die bereits eine langjährige Erfahrung im Bau von Robotern besitzen den Großteil der Elektronik auf einem Board platzieren. Die elektromagnetische Verträglichkeit (EMV) sollte in künftigen Projekten dieser Art ebenfalls nicht vernachlässigt werden. Für den Testaufbau eines Teilsystems des Roboters hingegen ist das modulare System V2.0 gut geeignet, da die benötigten elektronischen Komponenten schnell verbunden und programmiert werden können.

Eine Trennung der Energieversorgung von der Elektronik wäre für die Stabilität des Systems vorteilhaft. Ebenso würde eine Datenverbindung zwischen den Mikrocontrollern nach einem Industriestandard wie etwa RS485 eine störungsfreie Kommunikation gewährleisten.

Für die etwaige serielle Kommunikation mit den Beacons und dem PC wäre ein ZigBee¹⁷-Funkmodul der Firma Digi (ehemals MaxStream) denkbar, da die momentan eingesetzten Easy-Radio¹⁸ Funkmodule mit größeren Datenmengen nicht zurechtkommen und für das debuggen ungeeignet sind.

¹⁷ [digi, 2010]

¹⁸ [Easy-Radio, 2010]

8 LITERATUR

8.1 Bücher

[G.Schmitt, 2006] Schmitt, Günter: Mikrocomputertechnik mit Controllern der Atmel AVR-RISC-Familie, Hrsg. Oldenbourg Wissenschaftsverlag GmbH

[Lämmel, Cleve, 2004] Lämmel, Uwe; Cleve, Jürgen: Künstliche Intelligenz, Hrsg. Carl Hanser Verlag München Wien

8.2 Internetquellen

[Austrobot, 2010] <http://austrobot.info/>, Stand vom 01.03.2010

[digi, 2010] <http://www.digi.com/products/wireless/zigbee-mesh/xbee-zb-module.jsp#overview>,
Stand vom 08.06.2010

[Easy-Radio, 2010] <http://www.roboter-teile.de/Shop/index.php>, Stand vom 01.03.2010

[Robo-Racing-Team, 2010] <http://rrt.fh-wels.at/>, Stand vom 01.03.2010

[Regeln Eurobot, 2010] [http://www.planete-sciences.org/robot/data/file/coupe/2010/
E2010_rules_and_drawing_EN.pdf](http://www.planete-sciences.org/robot/data/file/coupe/2010/E2010_rules_and_drawing_EN.pdf), Stand vom 22.09.2009

8.3 Relevante Bachelorarbeiten

Die angeführten Bachelorarbeiten sind unter folgendem Link veröffentlicht:

<http://rrt.fh-wels.at/publications.html>

[Muckenhumer, 2010] Anbindung und Adaptierung einer mikrocontrollergesteuerten Kamera an einen autonomen Roboter

[Pautzenberger, 2010] Konzeptionierung, Konstruktion und Aufbau der Mechanik eines mobilen autonomen Roboters

[Rathgeber, 2010] Implementierung einer Armkinematik am Roboter für die Eurobot 2010

[Weinberger, 2010] Konzept und Programmierung des Mappings für autonome mobile Roboter

[Zauner, 2009] Modulares und skalierbares elektronisches System für einen Roboter

8 ANHANG

8.1 C-Code der KI

```

//*****
//Berechnung der nächsten Position
//*****
case KI_STRATEGY:

//1. Mögliche Anfahrtspositionen ermitteln
//2. Route zum Tor planen
//3. Wenn nötig ein Korrekturfahrt einleiten
//4. Erste Position anfahren
//5. Wenn während der Fahrt die Gegnererkennung oder der Endschalter des SensBords angesprochen hat
// dann zum alten Punkt zurückfahren

if((READ_FROM_MAILBOX(DRIVE_TASKNBR, STATUS) == ERROR_DRIVE) ||
(READ_FROM_MAILBOX(OBSERVER_TASKNBR, STATUS) == ERROR_END_BOARD) ||
(READ_FROM_MAILBOX(OBSERVER_TASKNBR, STATUS) == ERROR_END_ROBOT) )
{
ucStateKI = OBSTACLE_DETECTED;
}

//*****
switch(ucStateKI)
{
//*****
//Mögliche Anfahrtspositionen ermitteln
//*****
case POSSIBLE_POS:

//Schleifenindexe für den Scanrahmen ermitteln
ucStartIndexCol = ActRobotPos.uiXposition;
ucStartIndexRow = ActRobotPos.uiYposition;
ucStopIndexCol = ActRobotPos.uiXposition;
ucStopIndexRow = ActRobotPos.uiYposition;

if(ucFirstRun == 1)
{
OldRobotPos = RoutPoints[0];
ucFirstRun = 0;
}

ucCountPossiblePos = 0;
ucOptimalPoint = 0;

//Scanrahmen ermitteln
if(ucStartIndexCol > X0)
{
ucStartIndexCol--;
}
if(ucStartIndexRow > Y0)
{
ucStartIndexRow--;
}
if(ucStopIndexCol < X6)
{
ucStopIndexCol++;
}
}
}

```

```

if(ucStopIndexRow < Y7)
{
    ucStopIndexRow++;
}

//Rahmen abfragen und mögliche Anfahrtspositionen ermitteln
for(ucIndexCol = ucStartIndexCol; ucIndexCol <= ucStopIndexCol; ucIndexCol++)
{
    for(ucIndexRow = ucStartIndexRow; ucIndexRow <= ucStopIndexRow ; ucIndexRow++)
    {
        //Wenn es sich um kein FAKECORN und nicht um die Aktuelle Roboterposition handelt
        if((POI[ucIndexCol][ucIndexRow].ucElementName < FAKECORN))
            //(ucIndexCol != EnemyRobot.ucPOI_COL) && (ucIndexRow != EnemyRobot.ucPOI_ROW))
            {
                if((ucIndexCol != ActRobotPos.uiXposition) || (ucIndexRow != ActRobotPos.uiYposition))
                {
                    PossiblePositions[ucCountPossiblePos].uiXposition = ucIndexCol;
                    PossiblePositions[ucCountPossiblePos].uiYposition = ucIndexRow;
                    PossiblePositions[ucCountPossiblePos].ucElementName = POI[ucIndexCol][ucIndexRow].ucElementName;
                    ucCountPossiblePos++;
                }
            }
    }
}

//*****
//Auswertung der möglichen Anfahrtspositionen je nach Situation (Punkte, Spielzeit, Gegner, ...)
//*****
case ANALYSE_POS:

    ucMaxPointPos = FAKECORN;

    //Wenn die Tanks annähernd voll sind oder die Spielzeit 70s überschreitet
    if(((READ_FROM_MAILBOX(GREIFER_RECHTS_TASKNBR, ELEMENTCOUNTER) >= 3) &&
        (READ_FROM_MAILBOX(GREIFER_LINKS_TASKNBR, ELEMENTCOUNTER) >= 3)) ||
        (ucTimeToPlay <= 70)
        )
    {
        ucPriority = DRIVE_TO_GOAL;
    }
    else
    {
        ucPriority = MAX_POINTS;
    }

    //Priore Richtung je nach Spielfarbe ermitteln
    if(ucPlayColour == BLUE)
    {
        if(ActRobotPos.uiXposition > X0)
        {
            ucPriorDirectionX = ActRobotPos.uiXposition - 1;
        }
        if(ActRobotPos.uiYposition < Y7)
        {
            ucPriorDirectionY = ActRobotPos.uiYposition + 1;
        }
    }

    GoalReached.uiXposition = X0;
    GoalReached.uiYposition = Y7;
}
else if(ucPlayColour == YELLOW)
{
    if(ActRobotPos.uiXposition < X6)
    {
        ucPriorDirectionX = ActRobotPos.uiXposition + 1;
    }
    if(ActRobotPos.uiYposition < Y7)
    {
        ucPriorDirectionY = ActRobotPos.uiYposition + 1;
    }

    GoalReached.uiXposition = X6;
    GoalReached.uiYposition = Y7;
}
}

```

```

//Optimalen Anfahrtpunkt berechnen
if(ucPriority == MAX_POINTS)
{
    for(ucIndexMaxPoints = 0; ucIndexMaxPoints < ucCountPossiblePos; ucIndexMaxPoints++)
    {
        if(PossiblePositions[ucIndexMaxPoints].ucElementName <= ucMaxPointPos)
        {
            //Wenn die priore Richtung möglich ist
            if((PossiblePositions[ucIndexMaxPoints].uiYposition == ucPriorDirectionY) &&
                (PossiblePositions[ucIndexMaxPoints].uiXposition == ucPriorDirectionX) )
            {
                ucIndexRowOI = PossiblePositions[ucIndexMaxPoints].uiYposition;
                ucIndexColOI = PossiblePositions[ucIndexMaxPoints].uiXposition;
                ucOptimalPoint = 1;
            }
            //Wenn nur die priore X - Richtung möglich ist
            else if((PossiblePositions[ucIndexMaxPoints].uiXposition == ucPriorDirectionX) && (ucOptimalPoint > 2))
            {
                ucIndexRowOI = PossiblePositions[ucIndexMaxPoints].uiYposition;
                ucIndexColOI = PossiblePositions[ucIndexMaxPoints].uiXposition;
                ucOptimalPoint = 2;
            }
            //Wenn nur die priore Y - Richtung möglich ist
            else if((PossiblePositions[ucIndexMaxPoints].uiYposition == ucPriorDirectionY) && (ucOptimalPoint > 3))
            {
                ucIndexRowOI = PossiblePositions[ucIndexMaxPoints].uiYposition;
                ucIndexColOI = PossiblePositions[ucIndexMaxPoints].uiXposition;
                ucOptimalPoint = 3;
            }
            //Da Punkt möglich ist
            else
            {
                ucIndexRowOI = PossiblePositions[ucIndexMaxPoints].uiYposition;
                ucIndexColOI = PossiblePositions[ucIndexMaxPoints].uiXposition;
                ucOptimalPoint = 4;
            }

            ucMaxPointPos = PossiblePositions[ucIndexMaxPoints].ucElementName;
            ActRobotPos.uiXposition = ucIndexColOI;
            ActRobotPos.uiYposition = ucIndexRowOI;
        }
        else if((PossiblePositions[ucIndexMaxPoints].uiXposition == GoalReached.uiXposition) &&
            (PossiblePositions[ucIndexMaxPoints].uiYposition == GoalReached.uiYposition) )
        {
            ucIndexRowOI = PossiblePositions[ucIndexMaxPoints].uiYposition;
            ucIndexColOI = PossiblePositions[ucIndexMaxPoints].uiXposition;
            ActRobotPos.uiXposition = ucIndexColOI;
            ActRobotPos.uiYposition = ucIndexRowOI;
            ucOptimalPoint = 1;
        }
    }

    RoutPoints[ucRouteIndex].uiXposition = ucIndexColOI;
    RoutPoints[ucRouteIndex].uiYposition = ucIndexRowOI;
    ucRouteIndex++;
}

```

```

//Auf schnellstem Weg zum Ziel fahren
else if(ucPriority == DRIVE_TO_GOAL)
{
    for(ucIndexMaxPoints = 0; ucIndexMaxPoints < ucCountPossiblePos; ucIndexMaxPoints++)
    {
        //Wenn die priore Richtung möglich ist
        if((PossiblePositions[ucIndexMaxPoints].uiYposition == ucPriorDirectionY) &&
            (PossiblePositions[ucIndexMaxPoints].uiXposition == ucPriorDirectionX) )
        {
            ucIndexRowOI = PossiblePositions[ucIndexMaxPoints].uiYposition;
            ucIndexColOI = PossiblePositions[ucIndexMaxPoints].uiXposition;
            ucOptimalPoint = 1;
        }
        //Wenn nur die priore Y - Richtung möglich ist
        else if((PossiblePositions[ucIndexMaxPoints].uiXposition == ucPriorDirectionX) && (ucOptimalPoint > 2))
        {
            ucIndexRowOI = PossiblePositions[ucIndexMaxPoints].uiYposition;
            ucIndexColOI = PossiblePositions[ucIndexMaxPoints].uiXposition;
            ucOptimalPoint = 2;
        }
        //Wenn nur die priore X - Richtung möglich ist
        else if((PossiblePositions[ucIndexMaxPoints].uiYposition == ucPriorDirectionY) && (ucOptimalPoint > 2))
        {
            ucIndexRowOI = PossiblePositions[ucIndexMaxPoints].uiYposition;
            ucIndexColOI = PossiblePositions[ucIndexMaxPoints].uiXposition;
            ucOptimalPoint = 3;
        }
    }

    //Da Punkt möglich ist
    else
    {
        ucIndexRowOI = PossiblePositions[ucIndexMaxPoints].uiYposition;
        ucIndexColOI = PossiblePositions[ucIndexMaxPoints].uiXposition;
        ucOptimalPoint = 4;
    }

    ActRobotPos.uiXposition = ucIndexColOI;
    ActRobotPos.uiYposition = ucIndexRowOI;

    RoutPoints[ucRouteIndex].uiXposition = ucIndexColOI;
    RoutPoints[ucRouteIndex].uiYposition = ucIndexRowOI;
    ucRouteIndex++;
}

//Abbrechen, wenn die Route berechnet worden ist
if((RoutPoints[ucRouteIndex-1].uiXposition == GoalReached.uiXposition) &&
    (RoutPoints[ucRouteIndex-1].uiYposition == GoalReached.uiYposition) )
{
    ucStateKI = OPTIMIZE_POINTS;
    //printf("X: %u Y: %u\r", ucIndexColOI, ucIndexRowOI);
    printf("GR\r");
    ucFirstRun = 1;
}

else
{
    ucStateKI = POSSIBLE_POS;
    ucStateControlManager = 203;
    //printf("X: %u Y: %u\r", ucIndexColOI, ucIndexRowOI);

    break;
}

//*****
//Punkte überscheifen, wenn möglich und Offsets aufschalten
//*****
case OPTIMIZE_POINTS:

    //Punkte überschleifen nicht aufgeführt

//*****
//Offsets der Positionen setzen und eventuell eine Korrekturfahrt durchführen
//*****
case CORRECTION_DRIVE:

    printf("OPX: %u Y: %u\r", OldRobotPos.uiXposition, OldRobotPos.uiYposition);

    ucStateKI = CORRECTION_DRIVE;

    siPhi = (signed int)uiGetPosition(CS_ANTRIEB, &XPos, &YPos);
    siPhi = siPhi - ((siPhi/3600)*3600);

```

```

if(siPhi < 0)
{
    siPhi += 3600;
}

//Offsetgenerierung
if(OldRobotPos.uiYposition > RoutPoints[0].uiYposition)
{
    siOffsetX = +150;
}
else if(OldRobotPos.uiYposition < RoutPoints[0].uiYposition)
{
    siOffsetX = -150;
}

if(OldRobotPos.uiXposition < RoutPoints[0].uiXposition)
{
    siOffsetY = -100;
}
else if(OldRobotPos.uiXposition > RoutPoints[0].uiXposition)
{
    siOffsetY = 100;
}

if((RoutPoints[0].uiXposition == GoalReached.uiXposition) &&
    (RoutPoints[0].uiYposition == GoalReached.uiYposition) )
{
    siOffsetX = 0;
    siOffsetY = 0;
}

//*****
//Eigentlichen Anfahrtpunkt übergeben
//*****
case DRIVE_TO_POS:

    ActRobotPos = RoutPoints[0];
    ucRouteIndex = 0;

    fDirection = DIR_FORWARD;

    OldPoint = NextPoint;
    NextPoint = POI[RoutPoints[0].uiXposition][RoutPoints[0].uiYposition];
    //NextPoint.uiXposition += siOffsetX;
    //NextPoint.uiYposition += siOffsetY;

    siOffsetX = 0;
    siOffsetY = 0;

    //printf("Px: %u, Py: %u \r", RoutPoints[0].uiXposition, RoutPoints[0].uiYposition);
    ucStateKI = POSSIBLE_POS;
    ucStateControlManager = 204;

    break;

```

```
//*****  
//Gegner wurde während der Fahrt erkannt  
//*****  
case OBSTACLE_DETECTED:  
  
printf("OD\r");  
  
NextPoint = OldPoint;  
ActRobotPos = OldRobotPos;  
ucStateControlManager = 204;  
fDirection = DIR_BACKWARD;  
ucStateKI = POSSIBLE_POS;  
  
if(READ_FROM_MAILBOX(OBSERVER_TASKNBR, STATUS) == ERROR_END_BOARD)  
{  
    POI[RoutPoints[0].uiXposition][RoutPoints[0].uiYposition].ucElementName = FAKECORN;  
    WRITE_TO_MAILBOX(OBSERVER_TASKNBR, STATUS, 0);  
}  
else if(READ_FROM_MAILBOX(OBSERVER_TASKNBR, STATUS) == ERROR_END_ROBOT)  
{  
    WRITE_TO_MAILBOX(OBSERVER_TASKNBR, STATUS, 0);  
}  
  
break;  
}
```