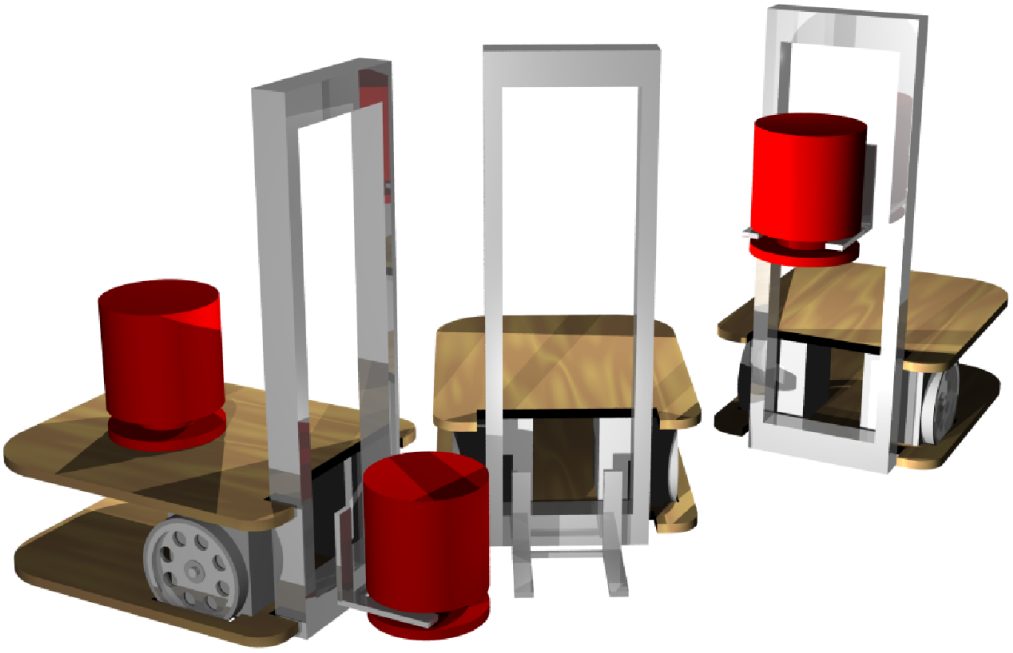
AG ROBOTERSYSTEME
FACHBEREICH INFORMATIK
AN DER UNIVERSITÄT KAISERSLAUTERN

Praktikum



Robotersystem der Gruppe A

Gruppe A

22. März 2007

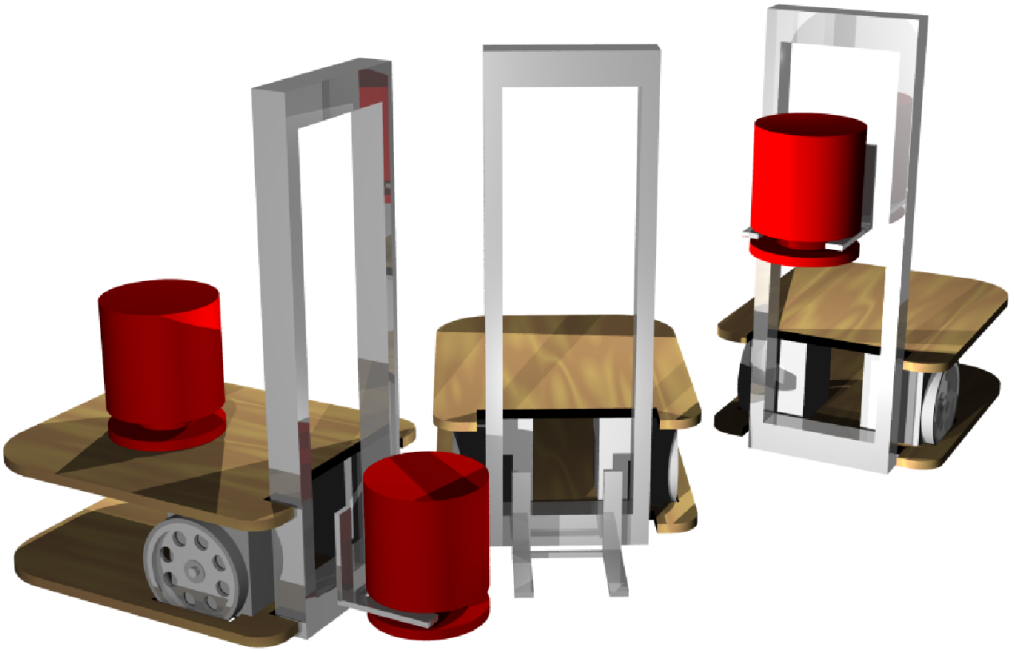
Inhaltsverzeichnis

1 Robotersystem der Gruppe A **3**
1.1 Aufbau des Roboters . . . 4
1.1.1 Geometrie der Grundplattform . . . 4
1.1.2 Infrarotsensoren . . . 4
1.1.3 Taster . . . 5
1.1.4 Kamera . . . 5
1.2 Die Karte . . . 6
1.2.1 Erstellen der absoluten Karte . . . 6
1.2.1.1 Pömpel . . . 7
1.2.1.2 Basen . . . 7
1.2.2 Darstellung der Karte . . . 7
1.2.2.1 Basen . . . 8
1.2.2.2 Pömpel . . . 8
1.2.2.3 Hindernisse . . . 8
1.2.2.4 Wegpunkte des A* Graphen . . . 8
1.2.2.5 Roboter . . . 8
1.3 Verhaltenssteuerung . . . 9
1.3.1 Zustände . . . 9
1.3.1.1 Idle . . . 9
1.3.1.2 Drive to LO . . . 9
1.3.1.3 Load LO . . . 9
1.3.1.4 Drive To Base . . . 9
1.3.1.5 Unload LO . . . 10
1.3.1.6 Push LO . . . 10
1.3.1.7 Search LO . . . 10
1.4 Basissteuerung . . . 11
1.4.1 Bewertung . . . 12
1.4.1.1 Anzahl der fehlgeschlagenen Anfahrversuche . . . 12
1.4.1.2 Alter des Lastobjektes . . . 12
1.4.1.3 Entfernung vom eigenen Roboter . . . 12
1.4.1.4 Nähe zum Gegner . . . 12
1.5 Fahrsteuerung und Strategien . . . 13
1.5.1 Routenplanung . . . 13
1.5.2 Bahnsteuerung . . . 16
1.5.3 Vor- und Nachteile . . . 16

1.6 Umwelterkennung 18
1.6.1 Hardware 18
1.6.2 Abstandsberechnung 19
1.6.3 Lastobjekt-, Basen- und Gegnererkennung 20
1.6.4 Wanderkennung 21
1.6.5 Optimierungen 23
1.6.6 Elemenierung der Außenregionen 23
1.6.6.1 Erkennung der Seitenwände 23
1.6.6.2 Ausblenden der Außenregionen 24
1.7 Ergebnisse und Diskussion 25

Literaturverzeichnis **26**

1. Robotersystem der Gruppe A



Robert Becker, Patrick Fleischmann, Luc Heischbourg, Jonas Mitschang, Thomas Pfister, Marcel Zimmer

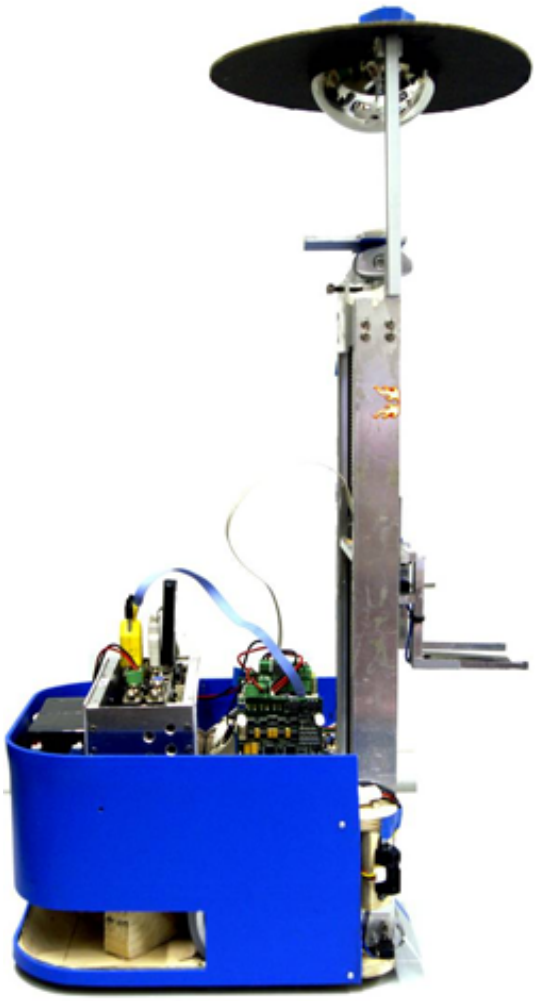


Abbildung 1.1: Der Profi

1.1 Aufbau des Roboters

1.1.1 Geometrie der Grundplattform

Die Grundplattform bilden zwei Sperrholzplatten an denen die Motoren, die Infrarotsensoren, die Gabel und die Taster angebracht sind. Um die Beweglichkeit des Roboters zu verbessern, wurden die beiden Platten um jeweils 5 cm gekürzt. Außerdem wurde ein Teil der oberen Plattform verlagert, um einerseits den Mini-PC und die Akkus besser unterzubringen, aber auch andererseits den Schwerpunkt des Roboters nach unten zu verlagern. (Siehe Abbildung 1.1)

1.1.2 Infrarotsensoren

Die zwei Short-Range IR-Sensoren und die vier Long-Range IR-Sensoren sind wie in Abbildung 1.2 angeordnet. Zur Hinderniserkennung beim nach vorne Fahren sind zwei Short-Range IR-Sensoren an den vorderen Ecken in einem Winkel von 45 Grad und ein Long-Range IR-Sensoren zentral angeordnet. Im Heck des Roboters sind drei Long-Range IR-Sensoren angebracht und sind gleich ausgerichtet wie die Frontsensoren.

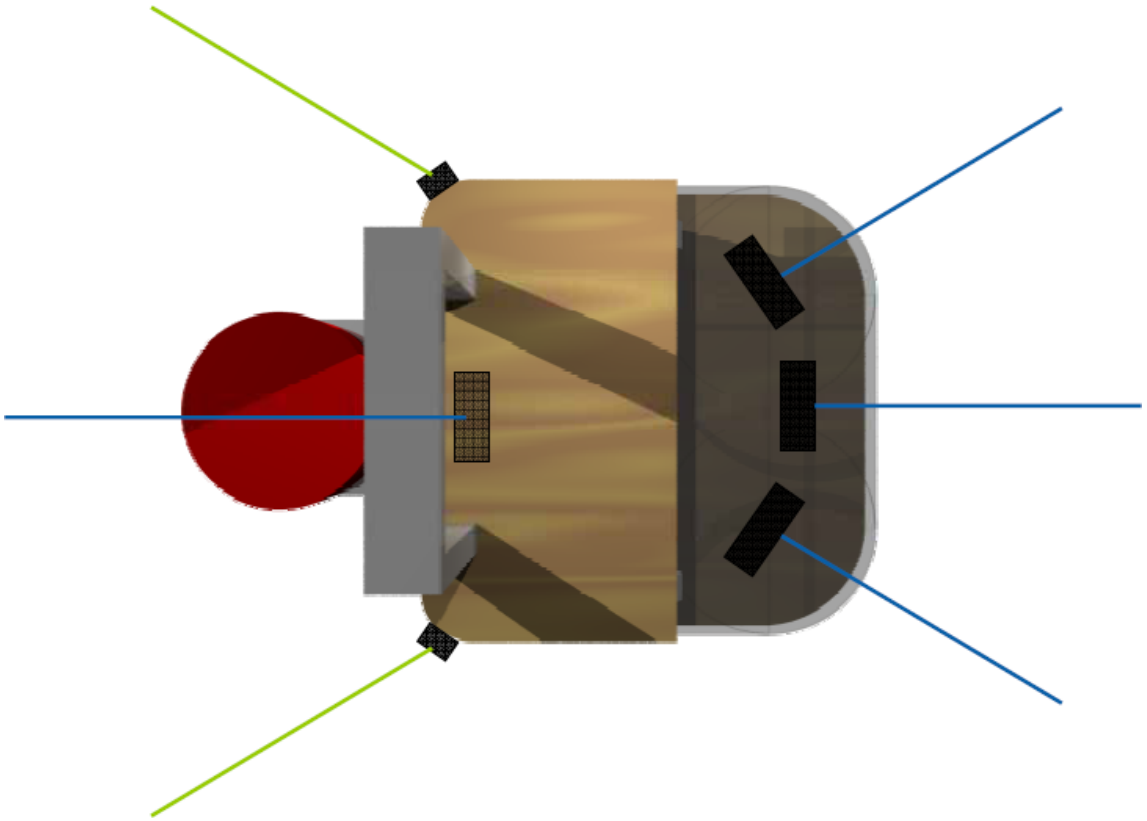


Abbildung 1.2: Short-Range IR-Sensoren (grün), Long-Range IR-Sensoren (blau)

Diese Anordnung der Infrarotsensoren hat sich bei den Testdurchläufen bewährt und auch beim Roboterwettbewerb traten keine Probleme auf.

1.1.3 Taster

Zwei Taster sind an der Front des Roboters angebracht. Wenn diese beim Anfahren an die Basis auslösen werden, wird der aufgeladene Pömpel abgesetzt. Die zwei restlichen Kontaktschalter wurden in der Gabel angebracht. Sie helfen beim Auf- und Abladen von Pömpeln.

1.1.4 Kamera

Da das Fahrverhalten des Profis grundsätzlich von der generierten Karte abhängt, müssen die Hindernisse, Basen und Pömpel so schnell wie möglich erkannt werden. Dies funktionniert am Besten mit einer Rundumsicht. Die Kamera wurde deshalb am oberen Ende des Gabelmastes angebracht und wie in Abbilung 1.11 zu erkennen, auf einen kugelförmigen Spiegel gerichtet.

1.2 Die Karte

Durch den erhöhten 360°-Rundumblick des Roboters wird ein Großteil aller Spielfeldobjekte gut und über einen längeren Zeitraum recht sicher erkannt. Das macht es möglich, alle diese Elemente in einer globalen Karte zu verwalten, anhand derer eine genauere Beurteilung der Umgebung möglich ist. Unter Verwendung solch einer Karte lassen sich auch komplexe Routen zwischen Hindernissen erstellen und souverän abfahren. Ebenso erlaubt Sie eine Bewertung der Lastobjekte anhand verschiedener Faktoren, was zu einem intelligenterem Einsammelverhalten führt. Allerdings ist diese Technik auch sehr anfällig, sobald die Odometrie ungenaue Positionsangaben liefert oder die Optik verzerrt ist.

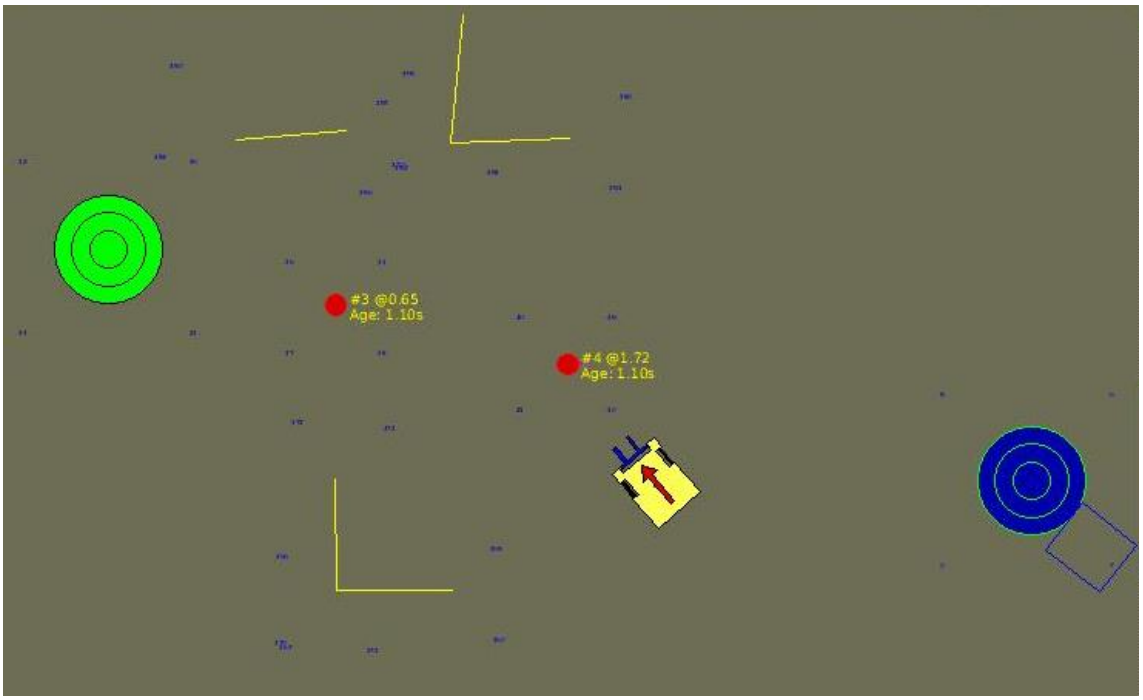


Abbildung 1.3: Darstellung der Karte in der GUI

1.2.1 Erstellen der absoluten Karte

Das Kameramodul liefert Polarkoordinaten der erkannten Objekte relativ zur aktuellen Position und Ausrichtung des Roboters. Diese müssen zur weiteren Verwendung zunächst in absolute kartesische Positionsdaten umgewandelt werden.

Dies geschieht, indem zuerst die Blickrichtung des Fahrzeugs zur relativen Ausrichtung des Objektes addiert wird. Anschließend werden diese neuen Polarkoordinaten in kartesische umgerechnet und letztens noch um die absolute Position des Roboters auf der Karte verschoben.

$$p_{abs} = (\alpha_{rel} + \alpha_{blickrichtung}, d).toCartesian() + p_{roboter}$$

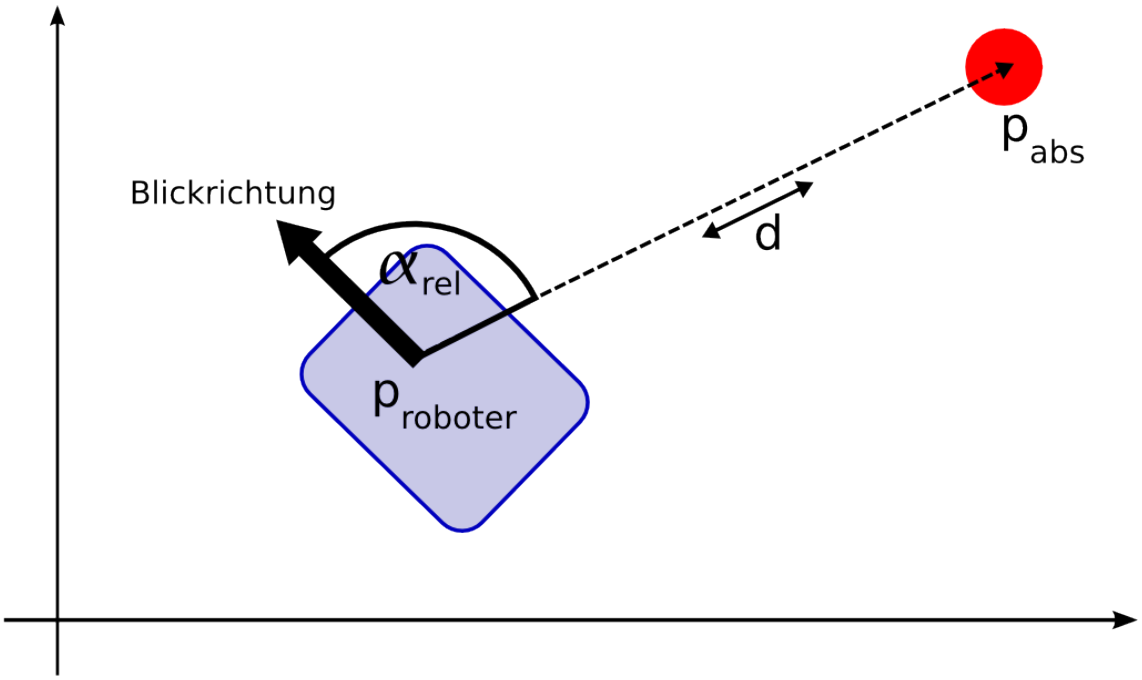


Abbildung 1.4: Umwandlung relativer Koordinaten in absolute

1.2.1.1 Pömpel

Anhand der errechneten Koordinaten wird versucht, Lastobjekte auf der bestehenden Karte wieder zu finden. Da die Bilderkennung jedoch gewissen Schwankungen unterliegt, werden die Pömpel in einem Radius von 20cm gesucht. Wird innerhalb dieses Kreises ein Lastobjekt wiederentdeckt, so wird dessen Lastseen-Zeitstempel aktualisiert. Überschreitet ein Objekt auf der Karte seine maximale Lebenszeit von 3 Sekunden, ohne wiedererkannt worden zu sein, so wird es verworfen. Pömpel auf der Gabel des Roboters oder auf den Basen werden ignoriert, da sie für das Verhalten keine Relevanz haben und so direkt auf einer niedrigen Ebene Fehler vermieden werden.

1.2.1.2 Basen

Das Wiederfinden der Basen gestaltet sich recht einfach, da diese bekanntermaßen recht weit auseinander liegen und aufgrund ihrer Größe gut vom Kamerasystem erkannt werden. Hier kommen im Grunde die gleichen Mechanismen wie bei den Lastobjekten zur Anwendung, nur erlauben die angesprochen Eigenschaften der Basen ein sehr großzügiges Umgehen mit den Toleranzparametern: der Wiedererkennungsradius beträgt einen Meter. Zusätzlich ist die Lebenszeit der gegnerischen Basis auf 15 Sekunden erhöht und die der eigenen Basis unbegrenzt, da deren Position essentiell für ein Korrektes Verhalten des Roboters ist.

1.2.2 Darstellung der Karte

Um das Verhalten des Roboters nachvollziehen zu können war es notwendig, die von ihm erstellte Karte in der GUI darstellen zu können. Dies wurde mithilfe des entsprechenden mca-gui Plugins und des Geometry Blackboards verwirklicht.

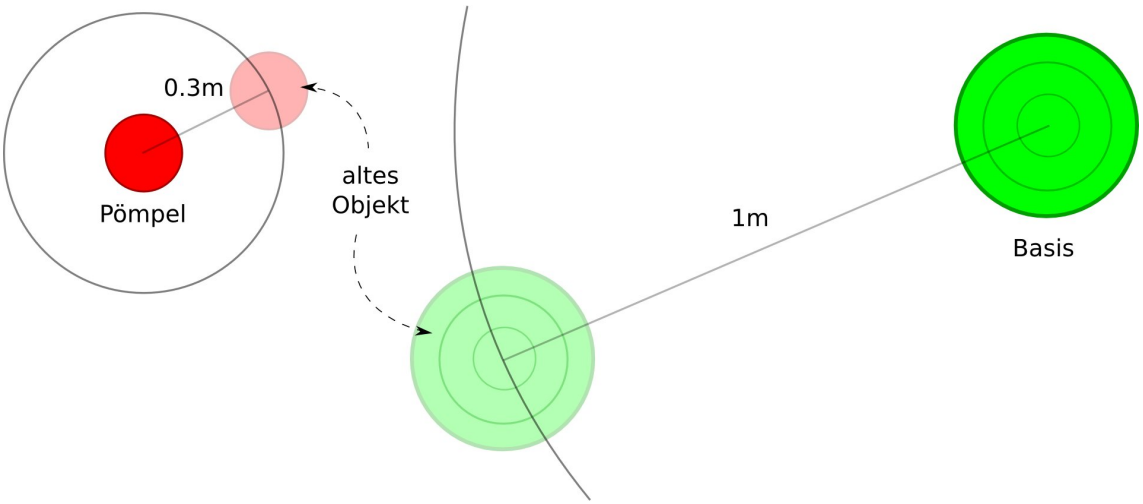


Abbildung 1.5: Wiedererkennungsradien

Die Kartenobjekte beinhalten jedoch noch weitere Informationen wie ihr Rating oder ihre Lebenszeit, die wir gerne als Text auf der Karte darstellen wollten. Da das vom MCA zu diesem Zeitpunkt jedoch noch nicht unterstützt wurde haben wir das Toolkit dementsprechend erweitert. Zusätzlich wurde von uns die Darstellung von Kreisen so modifiziert, dass diese bei der Darstellung in der GUI optional mit einer Farbe geüllt werden.

1.2.2.1 Basen

Die Basen werden als maßstabsgetreue Kreise visualisiert, wobei von deren wirklicher Form als Halbkreis abstrahiert wurde. Des weiteren erhalten sie eine unterschiedliche Einfärbung, um zwischen gegnerischer und eigener Basis unterscheiden zu können.

1.2.2.2 Pömpel

Zur Darstellung der Pömpel wurden gefüllte rote Kreise gewählt, neben denen die oben angesprochenen Informationen als Text erscheinen. Die verwendete Farbe geht immer weiter ins graue, je näher der Pömpel seiner maximalen Lebensdauer kommt, bis er letztendlich von der Karte verschwindet.

1.2.2.3 Hindernisse

Die Hinderniss werden als gelbe Linien gemäß ihrer reellen Lage auf die Karte gelegt.

1.2.2.4 Wegpunkte des A* Graphen

Analog werden bei Bedarf die Wege zwischen den A*-Wegpunkten als weiße Linien präsentiert.

1.2.2.5 Roboter

Die Darstellung des eigenen Roboters übernimmt die mca-gui anhand der über Sensorkanten übergebenen Positionsangaben.

1.3 Verhaltenssteuerung

Die MasterControl repräsentiert die eigentliche Verhaltenssteuerung des Fahrzeugs, die anhand von Zuständen realisiert wurde.

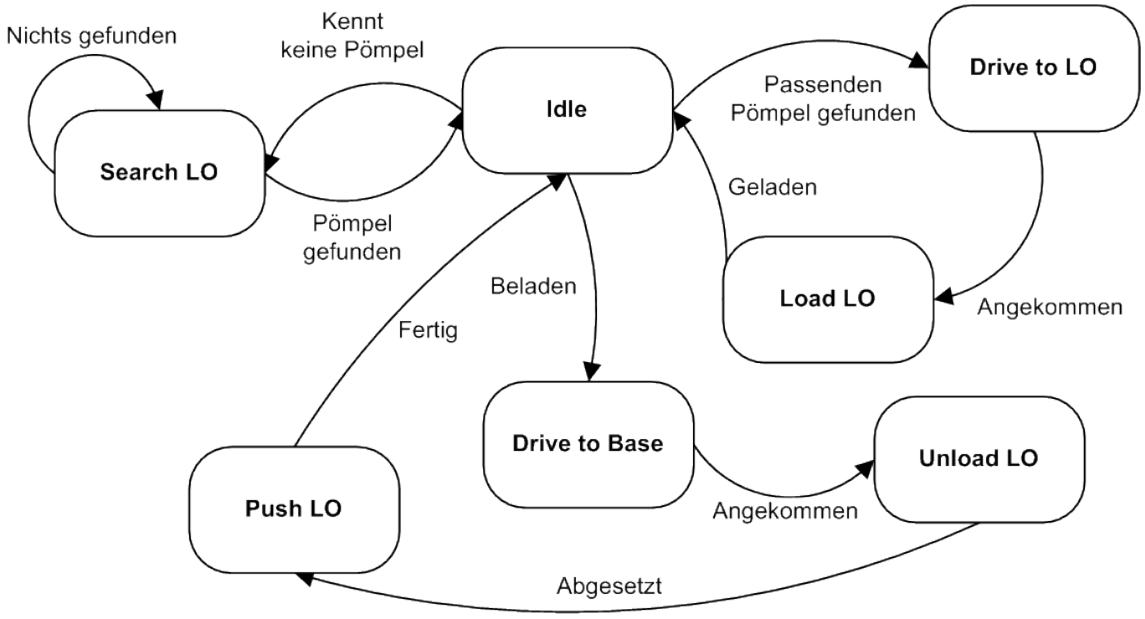


Abbildung 1.6: Vereinfachtes Zustandsdiagramm

1.3.1 Zustände

Die in Abb. 1.6 dargestellten Zustände repräsentieren jeweils bestimmte Aktionen, die der Roboter durchführt und nun im Folgenden erläutert werden.

1.3.1.1 Idle

Dies ist der Initialzustand des Robotors. Die Software untersucht die bekannte Umgebung und entscheidet dann, was als nächstes zu tun ist. Dieser Zustand wird immer aktiviert, sobald komplexere Abläufe abgeschlossen wurden oder nicht durchgeführt werden konnten.

1.3.1.2 Drive to LO

Anhand eines vom A*-Algorithmus gefundenen Weges wird ein vorher definierter Pömpel angefahren. Das ist immer der Pömpel mit der besten Bewertung.

1.3.1.3 Load LO

Das Fahrzeug wird auf das Lastobjekt ausgerichtet. Anschließend wird der Pömpel aufgegabelt, wobei die Ausrichtung des Fahrzeugs ständig korrigiert wird.

1.3.1.4 Drive To Base

Analog zu Drive To Lo wird die eigene Basis angefahren um Lastobjekte abzuladen.

1.3.1.5 Unload LO

Der Roboter richtet sich auf die eigene Basis aus und fährt solange vorwärts, bis der Frontbumper einen Kontakt meldet oder der Motorstrom stark ansteigt. Anschließend wird die Gabel abgesenkt und das Fahrzeug fährt 15cm rückwärts.

1.3.1.6 Push LO

Die Gabel wird auf eine Höhe gebracht, mit der alle Pömpel auf jeder Basishöhe geschoben werden können. Anschließend fährt der Profi vorwärts, bis der Motorstrom signalisiert, dass es nicht mehr weiter geht.

1.3.1.7 Search LO

Sind keine Lastobjekte mehr auf der Karte vermerkt, wird das Spielfeld auf der Suche nach neuen Lastobjekten abgefahren.

1.4 Basissteuerung

Das alles sind recht unterschiedliche Manöver, die auch jeweils unterschiedliche Parameter benötigen. Um das VehicleControlSystem (VCS) unseren Roboter entsprechend bewegen zu lassen haben wir die DriveModes eingeführt:

- DRIVE_MODE_STOP: Stop
- DRIVE_MODE_GOTO_OBJ: Fahre Objekt an
- DRIVE_MODE_GOTO_XY: Fahre Koordinaten an
- DRIVE_MODE_CUSTOM_DRIVE: Direkte Steuerung (Joystick)
- DRIVE_MODE_ROTATE: Drehe um relativen Winkel
- DRIVE_MODE_DISTANCE: Fahre bestimmte Strecke geradeaus
- DRIVE_MODE_ROTATE_TO_YAW: Drehe auf absoluten Winkel

Diese DriveModes besitzen jeweils eine interne ID, die dem VCS über eine Kante mitgeteilt wird. Auch die jeweiligen Parameter, wie beispielsweise die Strecke, die vom Profi in DRIVE_MODE_DISTANCE zurückgelegt werden soll, werden über eigene MCA-Kanten übertragen. Für eine genauere Erläuterung der einzelnen DriveModes siehe Kapitel Fahrsteuerung und Strategien.

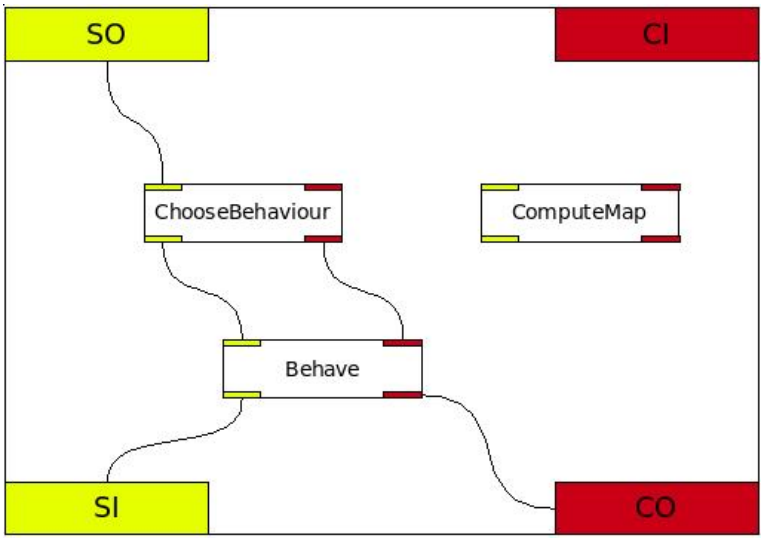


Abbildung 1.7: Mastercontrol MCA Gruppe

1.4.1 Bewertung

Wie schon im Kapitel Mapping erwähnt, erhalten die Pömpel eine Bewertung. Dies ermöglicht es uns beispielsweise Pömpel, die schon mehrfach fehlgeschlagene Anfahrversuche verursacht haben, vorerst zu ignorieren und erst gegen Spielende einzusammeln. In der Praxis hat sich gezeigt, dass zu diesem Zeitpunkt das Anfahren des Lastobjektes auch weniger Probleme verursacht, da sich zum einen der Anfahrwinkel geändert hat und andererseits auch weniger Hindernisse auf dem Spielfeld vorhanden sind.

Die letztendliche Bewertung eines Lastobjektes ergibt sich aus folgenden Faktoren:

1.4.1.1 Anzahl der fehlgeschlagenen Anfahrversuche

Jedes Mal, wenn der vom A* vorgeschlagene Weg aufgrund eines Eingreifens der Collisionsvermeidung nicht angefahren werden kann, wird das Rating des Pömpels dauerhaft verschlechtert. Wenn der A* Algorithmus zum ausgewählten Pömpel keinen Weg findet wird das Rating ebenfalls herabgesetzt.

1.4.1.2 Alter des Lastobjektes

Unter Umständen kommt es vor, dass die Kamera Pömpel erkennt, die eigentlich nicht existieren. Damit der Roboter nicht versucht, diese “Phantompömpel“ einzusammeln, erhalten alle Pömpel die nicht älter als 5 Sekunden sind einen pauschalen Malus auf ihre Bewertung. Wenn ein Pömpel über einen längeren Zeitraum sicher erkannt wird kann man davon ausgehen, dass es sich um ein echtes Lastobjekt handelt und der Malus wird nicht mehr eingerechnet.

1.4.1.3 Entfernung vom eigenen Roboter

Unserer Meinung nach macht es am meisten Sinn, zuerst die Pömpel einzusammeln, die am nähesten zum Roboter stehen. Deswegen geht die Entfernung zum Fahrzeug linear mit in die Bewertung ein.

1.4.1.4 Nähe zum Gegner

Da wir im Vorhinein Kollisionen mit dem Gegner vermeiden möchten, haben wir uns dazu entschlossen, innerhalb eines Radius von einem Meter zum gegnerischen Roboter das Rating für die Pömpel herabzusetzen.

1.5 Fahrsteuerung und Strategien

Das Vehicle-Control-System implementiert die grundlegende Fahrzeugsteuerung und die Wegfindung des Roboters. Zur Wegfindung bedient sich das VCS der globalen Karte des Moduls Mapping. Die gewünschte Zielposition erhält sie von dem Modul MasterControl. Außer den Fahrbefehl um zu einem bestimmten Punkt zu fahren implementiert das VCS noch einige weiter Fahrbefehle. Die möglichen Fahrbefehle des VCS sind:

- DRIVE_MODE_STOP: Der Roboter hält an.
- DRIVE_MODE_XY: Bei diesem Fahrbefehl wird auf Basis der Karte ein Weg zu einem Punkt auf der Karte geplant und abgefahren.
- DRIVE_MODE_OBJ: Dieser Fahrbefehl funktioniert ähnlich wie der DRIVE_MODE_XY, nur dass sich das VCS die Koordinaten des Zielpunktes selbst aus der Position des anzufahrenden Objekts bestimmt. Ein weiterer Unterschied zu dem DRIVE_MODE_XY ist, dass der Roboter vor dem Objekt anhält.
- DRIVE_MODE_DISTANCE: Der Roboter fährt eine gewisse Strecke geradeaus. Bei negativer Strecke fährt der Roboter rückwärts.
- DRIVE_MODE_ROTATE: Bei diesem Fahrbefehl dreht sich der Roboter auf der Stelle um einen angegebenen Winkel.
- DRIVE_MODE_ROTATE_TO_YAW: Wird dieser Fahrbefehl ausgeführt, dreht sich der Roboter zu einem absoluten Winkel auf der Karte.
- DRIVE_MODE_CUSTOM_DRIVE: Die Steuerung des Roboters wird direkt von der MasterControl übernommen.

Nur die Fahrbefehle DRIVE_MODE_XY und DRIVE_MODE_OBJ greifen auf die Routenplanung des VCS zurück, die restlichen Befehle sind eher primitiv und werden deshalb nicht weiter erklärt.

1.5.1 Routenplanung

Die Hauptaufgabe des VCS ist Planung eines Weges von seinem Standpunkt zu einem Zielpunkt und das Abfahren dieses geplanten Weges. Zur Planung des Weges bedient sich das VCS Standartmethoden der Robotik. Aus den Hindernissen, die in der Karte repräsentiert werden, wird ein Sichtgraph aufgebaut, in welchem der kürzeste Weg mittels des A*-Algorithmus bestimmt wird. Wird ein Weg gefunden, wird dieser von der Bahnsteuerung abgefahren.

Aufbau des Sichtbarkeitsgraphen

Der Sichtbarkeitsgraph dient als Grundlage zur Routenplanung des VCS. Der Graph repräsentiert mögliche Fahrwege des Roboters in der globalen Karte. Zum Aufbau des Sichtbarkeitsgraphen wird der Roboter auf einen Punkt reduziert. Dieser Punkt wurde so gewählt, dass er in der Mitte der zwei Räder liegt. Die Hindernisse wurden um den Raum erweitert, der vom Roboter nicht befahren werden darf. Desweiteren

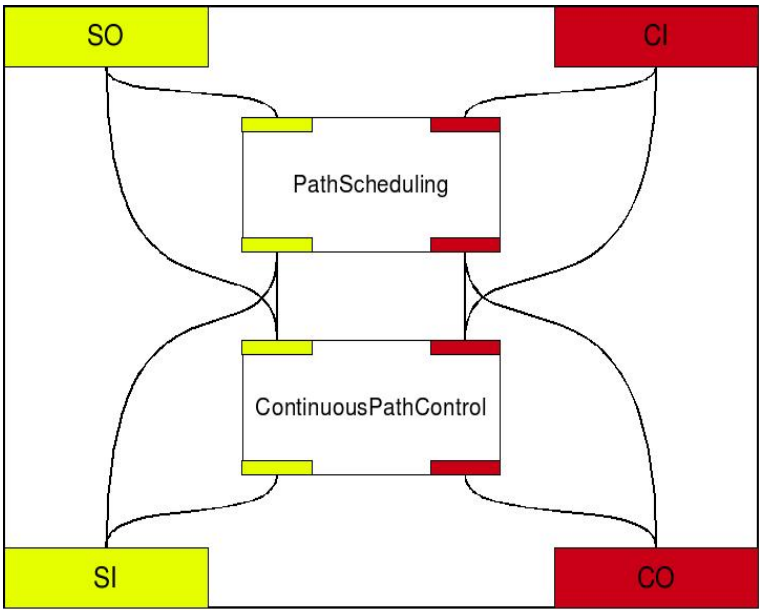


Abbildung 1.8: Aufbau des Modules VehicleControlSystem in MCA

wurde der Raum um die Hindernisse noch um einen Toleranzbereich erweitert, der eventuellen Ungenauigkeiten in der Karte vorbeugt. So sollen mögliche Kollisionen des Roboters mit den Hindernissen vermieden werden. Die entstehenden Hindernisse wurden der Einfachheit halber rechteckig gewählt. Die Hindernisse werden durch ihre vier Eckpunkte repräsentiert, welche auch gleichzeitig die Knoten für den Sichtgraphen bilden. Der Aufbau des Sichtgraph geschieht dann durch das Verbinden aller Knoten. Diese Verbindungen werden dann getestet, ob sie ein Hindernis schneiden oder sich einer der Knoten in einem Hindernis befindet. Ist dies der Fall werden die beiden Knoten nicht miteinander verbunden. Die benachbarten Knoten eines Knoten werden zusammen mit ihrem Abstand in einer Liste gespeichert. Diese Verbindungen bilden die Kanten des Graphen und die Abstände die Gewichte der Kanten. Als letzter Schritt zum Fertigstellen des Graphen werden noch je ein Knoten für den Start- und Zielpunkt eingefügt. Bei diesen Punkten ist zu beachten, dass sie auch innerhalb der erweiterten Hindernisse liegen können. Dies kann dann der Fall sein, wenn ein Lastobjekt nahe an einem Hindernis steht oder der Roboter zurück zur Basis fährt.

A*-Algorithmus

In dem erstellten gewichteten Graphen wird dann mittels des A*-Algorithmus der kürzeste Weg zu dem Zielpunkt bestimmt. Dieser Algorithmus stellt eine der Basismethoden bei der Suche nach dem kürzesten Weg in kantengewichteten Graphen dar. Der A*-Algorithmus basiert auf einer zweiteiligen Auswertefunktion für jeden Knoten. Die Auswertefunktion setzt sich aus der Wegstrecke zum Startknoten und einer Schätzfunktion für die zu erwartende kürzeste restliche Wegstrecke zusammen. Die zu erwartende kürzeste restliche Wegstrecke wurde als Abstand zum Zielpunkt gewählt. Diese Schätzfunktion ermöglicht die zielgerichtete Suche im Graphen, da zuerst die Knoten untersucht werden, welche sich näher am Zielknoten befinden.

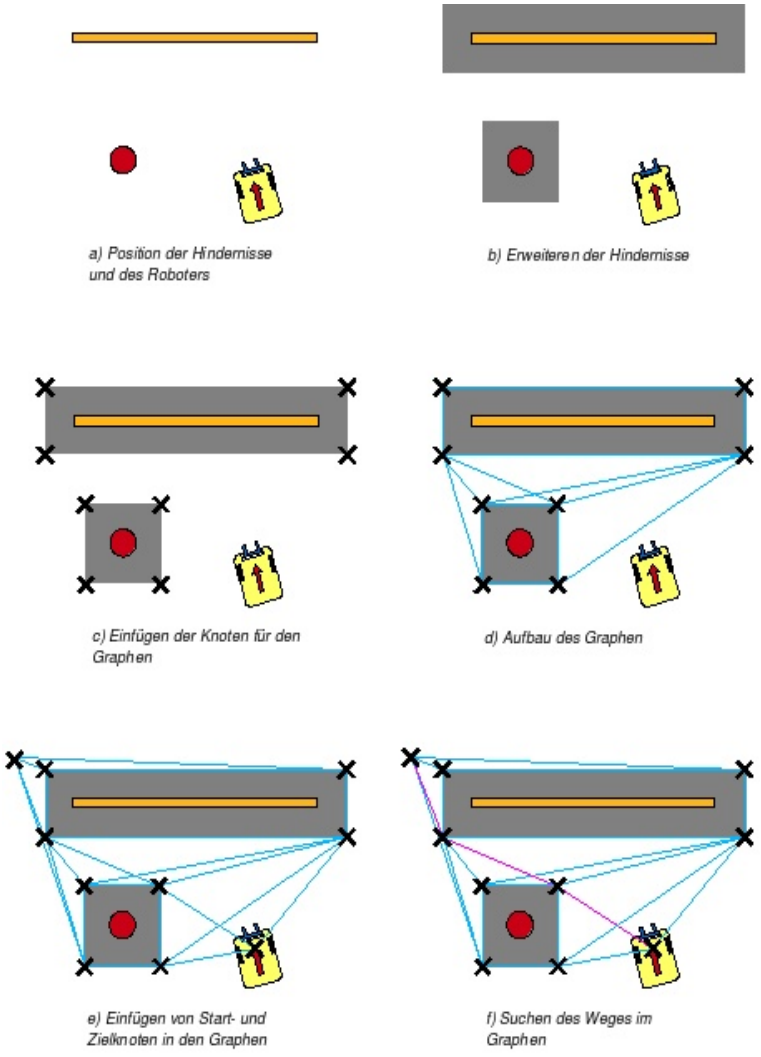


Abbildung 1.9: Aufbau des Sichtgraphen

Durch die zielgerichtete Suche ist ein effektiveres Verhalten zu erwarten als bei der nicht zielgerichteten Suche. Für den Fall, dass kein Weg zu dem Zielpunkt gefunden werden kann, wird die MasterControl hierüber in Kenntnis gesetzt.

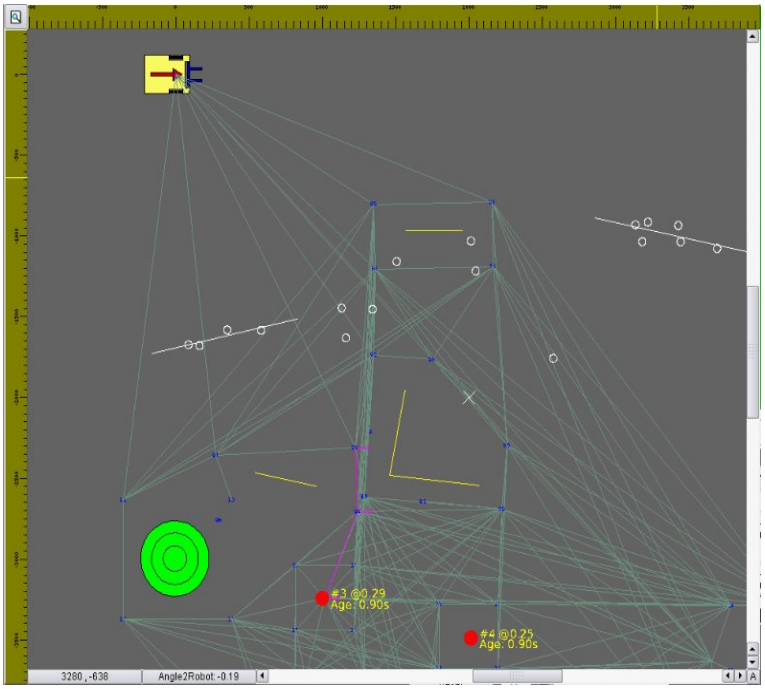


Abbildung 1.10: Sichtgraph in der MCA-GUI

1.5.2 Bahnsteuerung

Die Bahnsteuerung ist in dem Modul mContinousPathControl implementiert. Sie ist dafür verantwortlich, dass die geplante Route des A*-Algorithmus abgefahren wird. Der Bahnsteuerung sind zu einem Zeitpunkt nur die Koordinaten des nächsten Knotens/Wegpunktes bekannt. Sie versucht diesen Wegpunkt anzufahren, und sobald sie ihn als erreicht ansieht, verlangt sie nach dem nächsten Wegpunkt. Zum Anfahren der Wegpunkte richtet sich der Roboter zu dem Wegpunkt aus und fährt ihn direkt an. Während der Fahrt richtet sich der Roboter neu aus, falls der Winkel zwischen Fahrtrichtung und Richtung zu dem Wegpunkt zu groß wird.

1.5.3 Vor- und Nachteile

Der größte Vorteil unserer Fahrzeugsteuerung ist, dass die Routenplanung normalerweise den kürzesten Weg zu einem Zielpunkt findet. In der Praxis trat manchmal der Fall auf, dass der Roboter einen längeren Weg fuhr, was zu einem an dem Toleranzbereich um die Hindernisse und zum anderen an Ungenauigkeiten in der Karte lag. Diese beiden Faktoren führten dazu, dass manche Wege als nicht befahrbar berwertet wurden. Durch den größeren Toleranzbereich ergab sich zwangsweise ein etwas längerer Weg, da die Hindernisse mit einem gröserem Abstand umfahren wurden. Nützlich ist auch die Tatsache, dass die Wegfindung schon vor dem Losfahren erkennt, ob ein Zielpunkt erreichbar ist oder nicht. Hierdurch werden zuerst Lastobjekte angefahren, welche als erreichbar erkannt wurden. Ein weiterer Vorteil der Routenplanung

ist das Verhindern von Oszillationen. In der Praxis traten sie in Kombination mit der Kollisionsvermeidung leider trotzdem vereinzelt auf. Es zeigte sich auch, dass die Fahrzeugsteuerung nicht sehr geeignet für dynamische Umgebungen ist. So kann es passieren, dass der Roboter um Lastobjekte herumfährt die schon aufgesammelt worden sind oder das der gegnerische Roboter den geplanten Weg blockiert. Um das Verhalten bei diesen Problemen zu verbessern, wird der Graph alle 40cm neu berechnet und der Weg neu bestimmt. Dies ist auch vorteilhaft, da die Kamera auf kürzere Distanzen genauere Werte liefert als auf längere Distanzen.

1.6 Umwelterkennung

Zur Erkennung der Umwelt und zum Aufbau einer Umgebungskarte verwendet unser Roboter ausschließlich Informationen, die er vom Kamerasystem erhält. Die Infrarotsensoren werden nur zur lokalen Kollisionsvermeidung verwendet und in der Karte nicht berücksichtigt.

1.6.1 Hardware

Bei dem Aufbau des Kamerasystems haben wir uns für eine sogenannte Omnivision entschieden, worunter man eine Kamera mit Rundumsicht versteht. Die 360°-Sicht wird durch eine verspiegele Kugel erzielt, welche über der Kamera angebracht wird. Da uns eine solche Kugel nicht zur Verfügung stand, suchten wir nach Alternativen, darunter eine Christbaumkugel, ein Fahrradlichtreflektor und eine Suppenkelle. Letztendlich stellte sich die Suppenkelle aufgrund ihrer Form und ihrer Stabilität jedoch als bestes Objekt heraus. Des Weiteren entschieden wir uns die zur Verfügung gestellte Webcam in voller Auflösung, dass heißt mit 640x480 Pixel zu nutzen, da wir durch unsere Rundumsicht bereits eine geringere Auflösung hatten, als der nach vorn gerichtete Kameraansatz. Diese Auflösung erzeugte auf dem verwendeten Mini-PC allerdings eine zu hohe Last, sodass wir die Bildaufnahme auf ein Bild / Sekunde beschränken mussten.

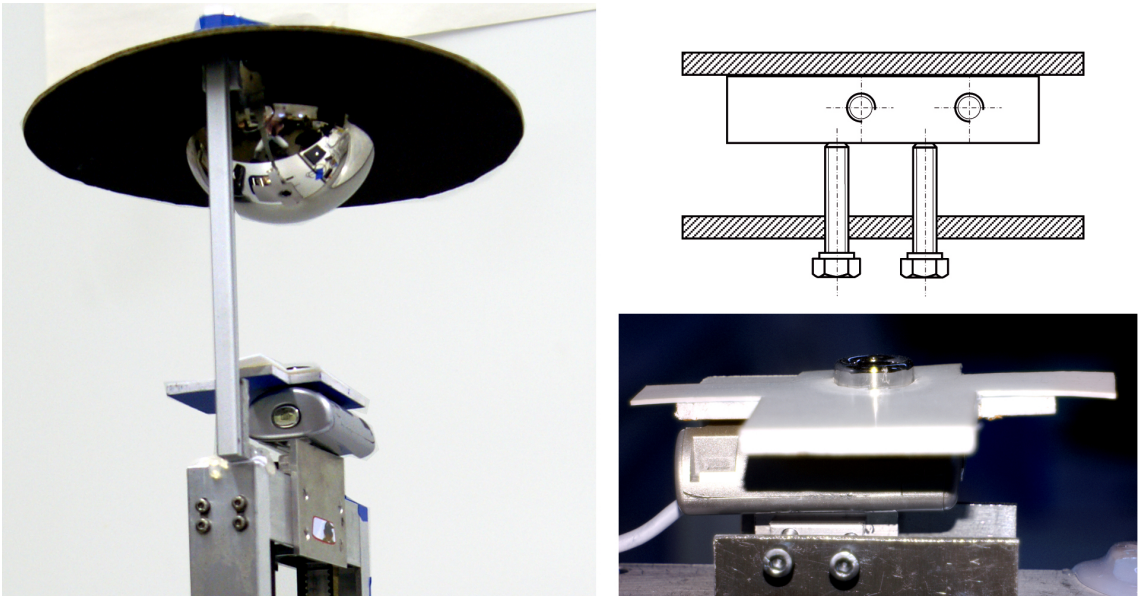


Abbildung 1.11: links: Realisierung der Omnivision, rechts: Kameraaufhängung

Ebenfalls erwähnenswert ist unsere Konstruktion zum automatisierten Weißabgleich, da beispielsweise Leuchtstoffröhren ein blaustichiges Licht erzeugen, wodurch die Farberkennung beeinflusst wird. Dieser Weißabgleich kann natürlich auch manuell durchgeführt werden, jedoch veränderten sich die Lichtverhältnisse auch innerhalb eines Spielfeldes durch Schattenwurf und ungleichmäßige Ausleuchtung. Um diesen Problemen zu entgehen, brachten wir eine weiße Plastikplatte an der Kamera an. Diese Fläche wurde vom Weißabgleich der Kameraelektronik automatisch auf weiß

geregelt, sodass wir uns in unserer Implementierung nicht um diese Farbstiche kümmern mussten. Die nächste Hürde bestand in der Befestigung der Kamera sowie der Ausrichtung des Kamerabildmittelpunktes unter dem Südpol der Kelle. Da sich diese Justierung als sehr anfällig herausstellte, konstruierten wir eine Aufhängung, die es möglich macht, die Kamera unter der fixierten Suppenkelle in alle Richtungen zu bewegen. Jedoch blieb die gesamte Konstruktion stoßanfällig und musste nach jedem Transport kontrolliert und gegebenenfalls neu justiert werden.

Die verwendete Omnivision hat natürlich den extremen Vorteil nicht nur nach vorne sehen zu können, sondern die Umwelterkennung um den gesamten Roboter auszudehnen. Damit kann ein großer Teil der Wegplanung und Kollisionsvermeidung aufgrund der Kameradaten erledigt werden. Natürlich hat diese Rundumsicht nicht nur Vorteile, jedoch überwiegen die Nachteiletrotz trotz ihrer Anzahl den oben genannten Vorteil nicht:

- Verzerrungen durch Verarbeitungsfehler / Verschmutzungen des Reflektors
- Reflektor nicht kugelförmig
- Geringere Auflösung (als nach vorne gerichteter Ansatz)
- Komplexere Berechnungen
- Ausrichtung, Befestigung, Justierung aufwändig und fehleranfällig

1.6.2 Abstandsberechnung

Wie bereits erwähnt handelte es sich bei unserem Reflektor nicht um eine perfekte Kugel, sodass eine direkte Umrechnung zu großen Fehlern führte. Um die Abstandsberechnung vom Roboter zu den erkannten Objekten zu verbessern, kam deshalb die Idee auf, für jeden Punkt des Kamerabildes bereits einen Abstandswert vorauszuberechnen, sodass zu Laufzeit der Distanzwert nur aus einer Tabelle gelesen werden muss. Dies verringert den Rechenaufwand zur Laufzeit, liefert bessere Ergebnisse als die lineare Interpolation zu Laufzeit und dass bei einem akzeptablem Speicherverbrauch von ca. 300 KB (307200 Byte, bei einer Darstellung der Abstandswerte als 8 Bit große Zahl).

Zur Erstellung der Abstandsmatrix haben wir zunächst ein Raster mit 25x25cm großen Quadraten auf dem Boden ausgebreitet. Zur Realisierung des Rasters haben wir rote Fäden verwendet, die auf dem Kamerabild gut zu erkennen waren. Das gespeicherte Bild importierten wir danach in ein JAVA-Programm, wo wir mit Hilfe von Mausklicks die Schnittpunkte des Rasters markierten und so den Pixelkoordinaten des Bildes Abstandswerte zuordneten. Das Resultat war eine Textdatei mit den aufgenommenen Daten, ein Liste von Punkten mit drei Dimensionen (x: Pixelkoordinate (Breite), y: Pixelkoordinate (Höhe), z: Abstandswert). Um jedoch einen Abstandswert für jeden Pixel des Bildes zu bekommen, muss man die Zwischenräume, für die keine Daten manuell erfasst wurden interpolieren. Hierzu verwendeten wir die kommerzielle Software Matlab um eine 3D-Interpolation durchzuführen. Das Ergebnis der Interpolation lässt sich aufgrund der 8 Bit großen Abstandswerten gut in einem Graustufenbild visualisieren, wie in der Abbildung 1.13 zu erkennen ist.

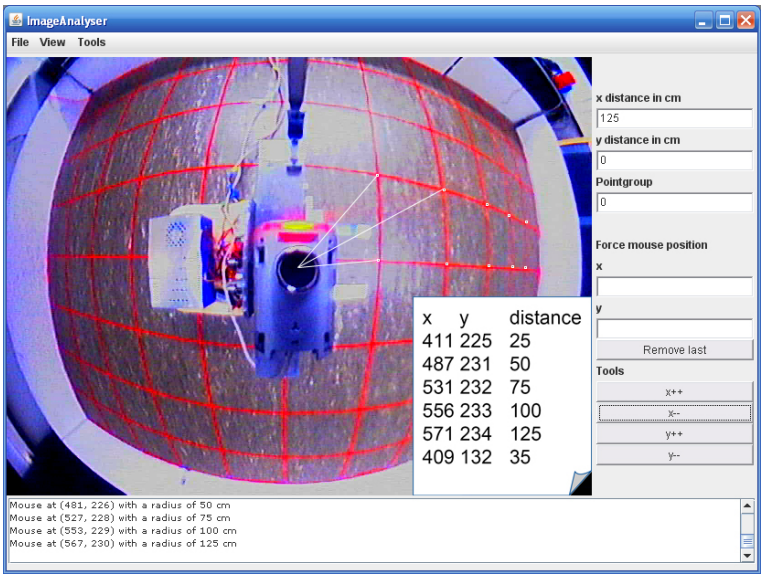


Abbildung 1.12: Erstellung der Abstandsmatrix mit Hilfe eines JAVA-Programms

1.6.3 Lastobjekt-, Basen- und Gegnererkennung

Zur Erkennung von Lastobjekten, Basen und des Gegners verwendeten wir die quelloffene Bibliothek CMVision. Die Bibliothek liefert die Koordinaten (bounding boxes) von Bereichen in denen zuvor definierte Farben vertreten sind. Diese bounding boxes sind zur Behandlung der Lastobjekte, Basen und Gegner ausreichend, da hier die genaue Form des Objektes für unsere Bahnplanungen nicht relevant war. Die erkannten Bereiche wurden dann in Form von Polarkoordinaten, dass heißt ein Winkel φ relativ zum Roboter und der Abstand vom Roboter zum Objekt, der mit Hilfe der Abstandsberechung ermittelt wurde, an die Karte übergeben.

Im Wesentlichen haben wir die Methoden von CMVision unverändert verwendet, jedoch zusätzliche Funktionen zur Optimierung implementiert, um z.B. zu kleine Objekte zu ignorieren und überschneidende Bereiche zu erkennen. Auch kam die Überlegung auf, CMVision so zu erweitern, dass man für eine Farbe mehrere Thresholds definieren kann, allerdings war dann das Definieren der Farbbereich sehr aufwändig, die Rechenzeit stieg an und das Ergebnis war nur unwesentlich besser.

```
[Colors]
(255,  0,  0) 0.80000 1 Red
(  0,  0,255) 0.990000 2 Blue
(  0,255,  0) 0.300000 3 Green
(  255,255,0) 0.990000 4 Yellow
(255,255,255) 1.00000 5 White
(0,255,255) 0.0 6 Robot

[Thresholds]
( 30:171, 100:135, 145:226)
( 30:150, 150:255, 0:120)
( 35:230, 100:140, 0:115)
```

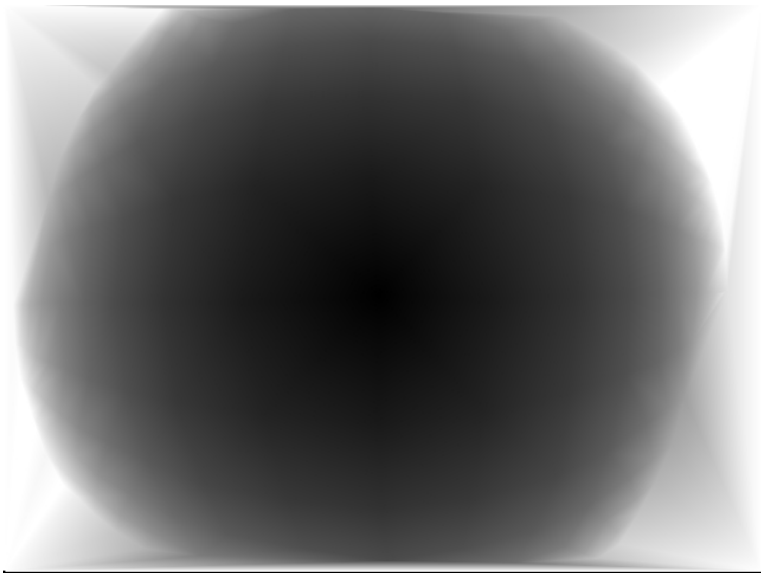


Abbildung 1.13: Die berechnete Abstandsmatrix visualisiert als Graustufenbild

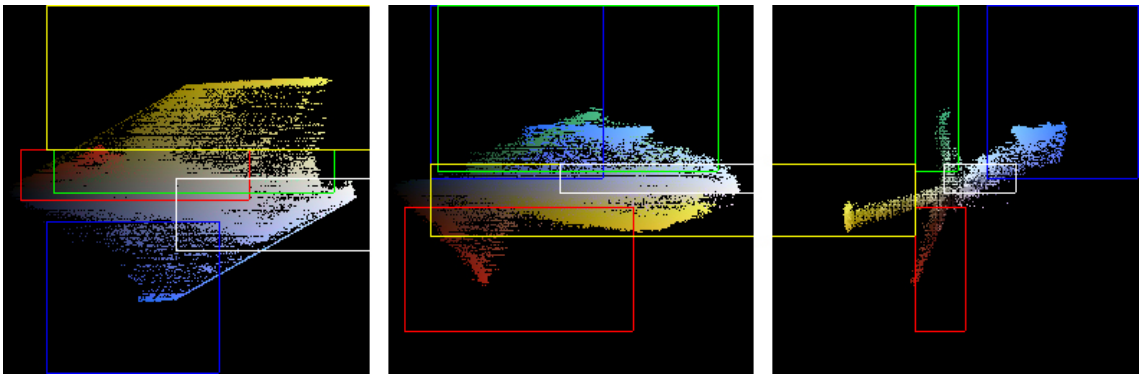


Abbildung 1.14: YU-, YV-, UV-Diagramm eines Bildes vom Spielfeld mit eingezeichneten Thresholds

```
( 30:255, 0:100, 110:160)
(175:255,120:170,110:130)
(128:128,128:128,128:128)
```

1.6.4 Wanderkennung

Zusätzlich zu den Lastobjekten, Basen und dem Gegner wurden auf dem Spielfeld Hindernisse in L- und H-Form platziert, bei denen eine Erkennung als bounding box nicht ausreichend ist, da man sich Fahrwege versperren würde und Objekte, die innerhalb des Hindernis stehen, nicht anfahren könnte.

Um diesem Problem zu begegnen, überlegten wir uns einen Algorithmus, den wir Strahlenalgorithmus nannten und dessen Ablauf in Abbildung 1.15 visualisiert ist. Die in Klammern stehenden Zahlen beziehen sich auf die Einzelbilder der Abbildung. Der Algorithmus bedient sich zunächst dem so genannten Blobimage, welches man

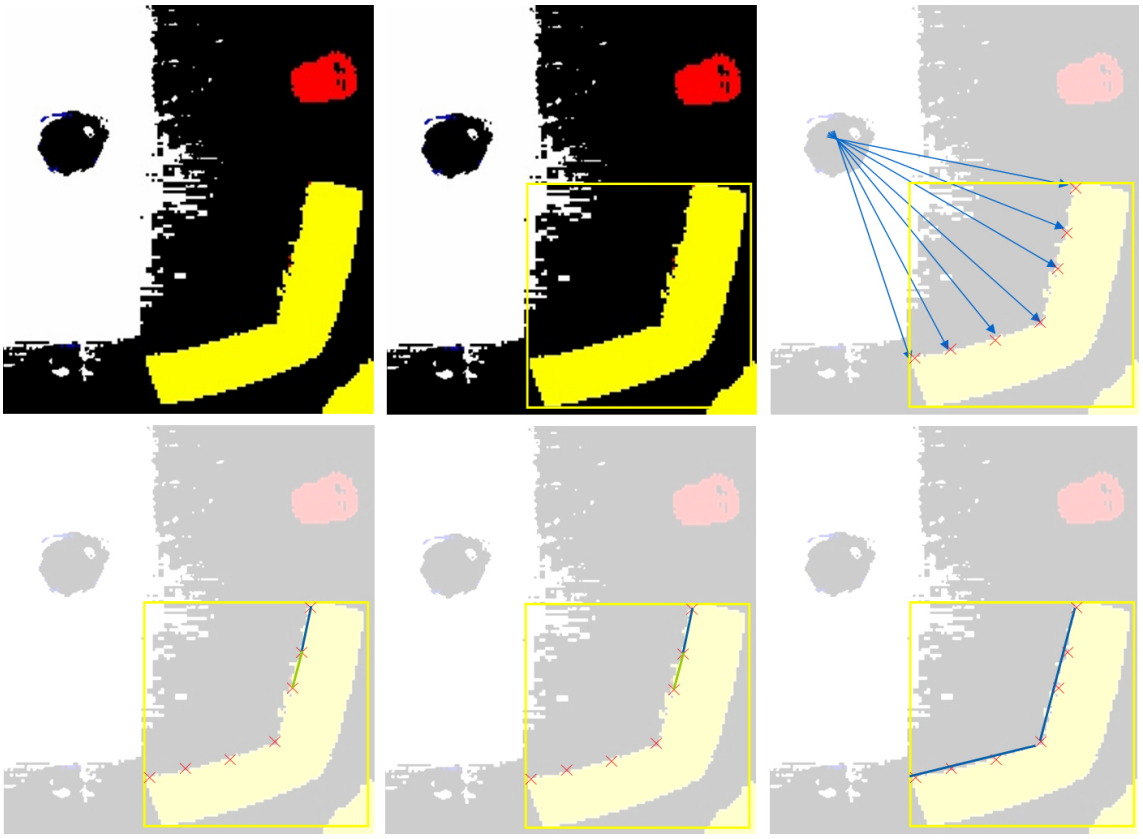


Abbildung 1.15: Visualisierung des Strahlenalgorithmus zur Erkennung der Wände

von CMVision durch `vision->testClassify()` erhält (1), sowie der von CMVision bestimmten bounding box (2). Danach werden innerhalb des erkannten Rechtecks virtuelle Strahlen verfolgt, die in einstellbaren Winkelabständen vom Kameramittelpunkt ausgesandt werden. Trifft der Strahl auf einen gelben Pixel, so wird dieser Punkt gespeichert und der Strahl abgebrochen (3). Anschließend werden die Punkte zu Liniensegmenten verbunden (4), wobei wiederum die Reduktion auf eine Linie das Ziel ist (5), solange ein bestimmter Winkel nicht unterschritten wird (6). Die gefundenen Linien werden in Polarkoordinaten an die Karte übergeben.

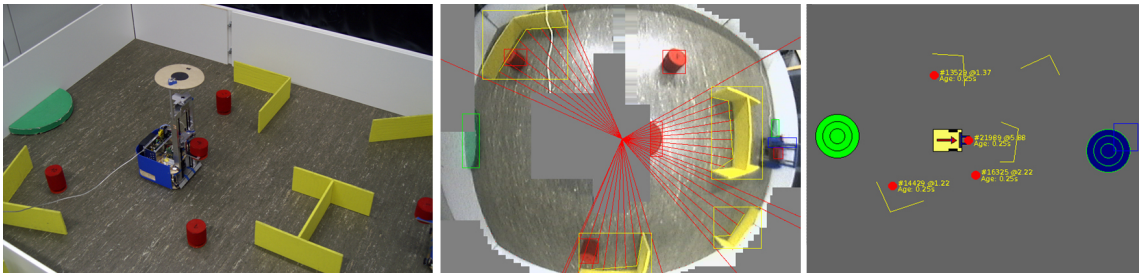


Abbildung 1.16: Bilderkennung in Aktion: links: Foto des Spielfeldes, mitte: Bild der Webcam, rechts: Darstellung in der Karte

1.6.5 Optimierungen

Neben den schon erwähnten Änderungen in CMVision sind zwei Maßnahmen unserer Bildverarbeitung besonders erwähnenswert. Wie im mittleren Webcambild der Abbildung 1.16 gut zu erkennen ist, sieht der Roboter weder sich selbst noch die Aufhängung des Reflektors. Die Aufhängung des Relektors wir ausgeblendet, indem wir die Randpixel duplizieren. Dies hat den Vorteil, dass Wände und andere Objekte, die durch den grauen Aluminiumstab der Aufhägung getrennt würden, als zusammenhängender Gegenstand erkannt werden. Unseren eigenen Roboter blenden wir aus dem Bild aus, um

- LEDs und sonstige farbige Bauteile, die sich auf auf dem Roboter befinden, von der Erkennung auszuschließen
- CMVision zu beschleunigen (da in den ausgeblendeten Bereichen keine Objekte erkannt werden, die man ohnehin verwerfen würde).

Wir arbeiten dabei destruktiv auf dem Originalbild, d.h. wir löschen die Bildinformationen an den Stellen, an denen wir den eigenen Roboter erkennen, indem wir die Bereiche auf eine neutrale Farbe setzen. Diesen Vorgang kann man sehr effizient mit der Methode `memset(...)` erledigen.

1.6.6 Elemenierung der Außenregionen

Der große Sichtbereich der Omnivision hat nicht nur Vorteile. Ein großer Nachteil ist, dass auch Gebiete außerhalb des Spielfeldes auf dem Kamerabild sichtbar sind. Befinden sich in diesen Regionen z.B. rote Gegenstände (Stühle, Pullover usw.) werden diese als Lastobjekte erkannt und angefahren. Um dem entgegen zu wirken haben wir uns überlegt wie wir alle Regionen außerhalb des Spielfeldes ausblenden können.

1.6.6.1 Erkennung der Seitenwände

Unsere erste Idee basierte darauf die weißen Seitenwände zu erkennen und als Hindernisse der Karte zu übergeben, sodass mögliche erkannte Lastobjekte außerhalb des Feldes nicht angefahren werden können. Hierzu erweiterten wir unseren Strahlenverfolgunsalgorithmus, sodass dieser die weißen Außenwände abtastet und aus den erhaltenen Punkten einen geschlossenen Polygonzug bildet. Das erste Problem das dabei auftrat, war dass die Lichtreflektionen am Boden teilweise kaum von den Wänden zu unterscheiden waren. Wir experimentierten mit verschiedenen Kameraparametern, aber entweder fehlten uns Teile der Wand, oder die Reflektionen waren zu stark sichtbar. Um die Außenwände dennoch relativ sicher zu detektieren, legten wir einige Regeln fest, wann ein ”weißer Fleck” Wand und wann Reflektion ist. Da Wände fast immer am Rand des Kamerabildes lagen, vergrößerten wir den Startradius unserer Strahlen um Reflektionen in Roboternähe von vornherein zu ignorieren. Da die Wände relativ viel Fläche auf dem Bild einnahmen, legten wir auch fest, dass ein ”Wand-Strahl” immer zunächst mindestens 6 aufeinanderfolgende schwarze Pixel und dann 10 aufeinanderfolgende weiße Pixel durchlaufen muss. Mit dieser Methode konnten wir die Außenwände relativ gut detektieren. Da die Umrechnung der Kameradaten in Entfernungen zum Bildrand hin jedoch massiv schlechter wird (Fehler $>$ 10cm/Pixel), ”sprangen” die Wandpunkte auf der Karte so stark, dass die Daten unbrauchbar waren.

1.6.6.2 Ausblenden der Außenregionen

Nachdem wir die Randpixel der Außenwände recht stabil detektieren konnten, überlegten wir uns eine andere Methode die störenden Außenregionen zu eleminieren. Ähnlich wie beim Ausblenden des Roboteraufbaus arbeiteten wir nach einem ersten Detektionsdurchgang erneut destruktiv auf das Kamerabild. Ausgehend von jedem erkannten Wandrandpunkt zeichneten wir ein graues Rechteck zum zugehörigen Bildeckpunkt. Alle Punkte zusammengenommen wurde so sehr effizient das Gebiet außerhalb des Spielfeldes grau eingefärbt.

1.7 Ergebnisse und Diskussion

Im Abschlusswettbewerb hat sich der Profi mit Bravur geschlagen. Die primären Aufgaben wie das Finden und Aufgabeln von Lastobjekten und diese zur Zielplattform transportieren, und Hindernisvermeidung, wurden alle erfüllt. Außerdem wurde noch das Sichern von eigenen Lastobjekten auf der Zielplattform implementiert. Alleine das Stapeln und Aufnehmen fremder Pömpel von der gegnerischen Zielplattform konnte wegen Zeitmangel nicht beachtet werden.

Außerdem ist noch hervorzuheben, dass das Praktikum trotz des vielen Zeitaufwand jede Menge Spaß gemacht und viel Wissenswertes über Robotik übermittelt hat. Angefangen hat es mit den Vorversuchen, die uns in die Materie eingeführt haben, da die wenigstens von uns schon mit einer derartigen Hardware gearbeitet haben. Desweiteren wurde uns kein fertiger im Handel gekaufter Robotergabelstapler vor die Nase gesetzt, den wir dann nur noch programmieren mussten. Den größten Teil des Roboters konnten wir verändern, und deshalb hinderte uns nichts dran die Grundplattform kleiner zu schneiden, und den Roboter tiefer zu legen. Wir konnten unseren Ideen freien Lauf lassen.

Zu guter Letzt noch ein Dank an die AG Robotersysteme, speziell den Betreuern des Praktikums, die immer ein offenes Ohr für unsere Probleme hatten.

Literaturverzeichnis