Institut für Informatik
Lehrstuhl für Programmierung und Softwaretechnik

LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Projektarbeit

Computergestützte visuelle Analyse eines Roboters

Stefan Meindl

Aufgabensteller: Prof. Dr. Martin Wirsing
Betreuer: M.Sc. Annabelle Klarl
Abgabetermin: 31.03.2012

Ich versichere hiermit eidesstattlich, dass ich die vorliegende Arbeit selbstständig angefertigt, alle Zitate als solche kenntlich gemacht sowie alle benutzten Quellen und Hilfsmittel angegeben habe.

München, den 31. März 2012. . . . . .

Kurzzusammenfassung:

Ziel dieser Arbeit ist es ein Framework bereit zu stellen, welches es ermöglicht Experimente mit mobilen Robotern per Videobild zu analysieren. Dabei wird darauf Wert gelegt, die Komponenten des Frameworks weitestgehend austauschbar zu gestalten, um eine möglichst große Flexibilität bei der Auswertung verschiedener Experimente zu erreichen.

Die Arbeit erläutert die Architekturentscheidungen, die dieses Ziel herbeiführen sollen und implementiert exemplarisch eine Realisierung der Framework-Komponenten. Diese Implementierung wird an einem Lego MINDSTORMS Roboter mit einer Linefollower-Implementierung erprobt.

Der Roboter folgt einem vorgegebenen Kurs. Der optimale Weg, den der Roboter im Kurs folgen kann, wird automatisch berechnet. Das Framework bestimmt die Position des Roboters mit Hilfe eines Markers. Das Framework liefert auf dieser Basis die Abweichung des Roboters zur Ideallinie und sammelt diese Daten zur anschließenden Analyse des Experiments.

Abstract:

The aim of this study is to provide a framework that allows to analyze experiments with mobile robots using video images. It is considered important to design the components of the framework largely interchangeably to achieve the greatest possible flexibility in the evaluation of various experiments.

The work explains the architectural decisions that are made to achieve this goal. An exemplary implementation of framework components is provided as a proof of concept. This implementation is tested on a Lego MINDSTORMS robot with a linefollower implementation.

The robot follows a set course. The optimal path that the robot can follow in the course is calculated automatically. The framework determines the position of the robot with the aid of a marker. The framework uses this information to compute the deviation from the ideal line and collects this data for subsequent analysis of the experiment.

Danksagung
Dankesworte

Inhaltsverzeichnis

1 Einleitung 1
1.1 Aufgabenbeschreibung 1
1.2 Outline 1
2 Verwandte Arbeiten 3
2.1 Forschungsroboter 3
2.1.1 Beispiele für Forschungsroboter 3
2.1.2 Lego MINDSTORMS NXT 4
2.2 Roboternavigation 5
2.2.1 Aktoren und Sensoren 5
2.2.2 Navigation (Typen von Aktoren & Sensoren zu Navigation) 5
2.2.3 Linefollower 6
2.3 Wegfindung 6
2.3.1 Algorithmen zur Routenberechnung 6
2.3.2 Der A*-Algorithmus 7
2.4 Bildverarbeitung und Markererkennung 7
2.4.1 Frameworks zur Markererkennung 8
2.4.2 ARToolKit 8
3 RoboTracker 11
3.1 Motivation 11
3.2 Funktionsumfang 11
3.2.1 Umgebungsanalyse 11
3.2.2 Berechnung der Ideallinie und Start-bzw. Zielpunkt 12
3.2.3 Aufzeichnung, Analyse und Auswertung der Daten 13
3.3 Architektur 14
3.3.1 Pakete 15
3.3.2 Programmsequenz 17
4 Experiment 19
4.1 Aufbau 19
4.1.1 Kurse 20
4.1.2 Roboter 20
4.1.3 Kamera 21
4.2 Durchführung 21

4.3 Ergebnis 22
4.4 Fazit 27
5 Konklusion und Ausblick 29
6 Literaturverzeichnis 31
7 Bedienungsanleitung 31

1 Einleitung

Schon seit jeher verwenden Menschen technische Errungenschaften, um Aufgaben zu erleichtern oder überhaupt erst lösbar zu machen – sie reichen vom einfachen Steinkeil bis zu hochkomplexen Industrierobotern oder Raumsonden, die eine Vielfalt von Funktionalitäten bereitstellen. Viele Herausforderungen sind heutzutage zu komplex oder gefährlich, um sie ohne technische Unterstützung lösen zu können. Roboter als programmierbare Maschinen bieten hier eine Möglichkeit diesen Problemen entgegen zu treten.

Heutzutage sind die Ansprüche und damit die Herausforderungen gewachsen. Roboter werden unlängst auch dort eingesetzt, wo menschliches Versagen ein Desaster auslösen könnte. Deshalb liegt eine besondere Bedeutung auf deren fehlerlosen Funktion und um diese sicher zu stellen, benötigt man Werkzeuge zur Analyse.

Modulare, programmierbare Roboter, wie die Lego MINDSTORMS, sind heute nicht nur ein beliebtes Spielzeug, sondern eignen sich auch hervorragend zu Forschungszwecken. Deshalb findet man sie an vielen Lehrstühlen, die sich mit Themen der Robotik auseinandersetzen.

Dabei ist es natürlich wichtig die Experimente mit den Robotern analysieren zu können. Ein eleganter Weg ist es dies ausschließlich anhand eines Videobildes zu bewerkstelligen.

Diese Arbeit bietet einen Lösung dieses Ansatzes.

1.1 Aufgabenbeschreibung

Diese Arbeit beschäftigt sich mit der videogestützten Analyse von autonomen, mobilen Robotern. Ziel ist es ein Framework zur Verfügung zu stellen, das es erlaubt aus Videodaten das Verhalten von Robotern mess– und dadurch vergleichbar zu machen.

Bei einem Experiment wird ein Roboter auf einer weißen Ebene mit einer aufgezeichneten, schwarzen Linie, beziehungsweise einem Rundkurs platziert. Der Roboter ist ein Linefollower und damit in der Lage dem Kurz eigenständig zu folgen.

Dabei wird besonderes Augenmerk auf die Abweichung des tatsächlichen Roboterkurses von der Ideallinie des Kurses gelegt. Um dieses Ziel zu erreichen wird der Roboter während er versucht dem Kurs zu folgen, mittels einer über dem Kurs installierten Kamera, aufgezeichnet. Die so gewonnen Informationen werden dann von dem Programm ausgewertet und zu verschiedenen Kennzahlen verdichtet, zum Beispiel der benötigten Gesamtzeit oder der durchschnittlichen Abweichung des Roboters von der Ideallinie.

1.2 Outline

Die Arbeit ist folgendermaßen gegliedert. Im Kapitel *verwandte Arbeiten* werden Konzepte und Technologien, die in der implementierten Lösung verwendet werden näher erläutert. Im Kapitel *Robotracker* wird detailliert auf das eigentliche Analysetool eingegangen. Aufbau, Durchführung und Ergebnis einer praktischen Anwendung der Lösung werden im Kapitel *Experiment* dargestellt. Der Schluss der Arbeit fasst die Ergebnisse der Projektarbeit zusammen und gibt Ausblick auf weitere zu untersuchende Ansätze.

2 Verwandte Arbeiten

In diesem Kapitel werden im Rahmen dieser Projektarbeit verwendete Konzepte, Technologien und Algorithmen näher erläutert.

2.1 Forschungsroboter

Forschungsroboter dienen sowohl der Vermittlung von Konzepten der Robotik, als auch der unkomplizierten Erprobung neuer Ansätze. Bei dieser Projektarbeit kommt ein aus Lego MINDSTORMS konstruierter Roboter zum Einsatz.

2.1.1 Beispiele für Forschungsroboter

Der Anspruch an Forschungsroboter ist hohe Flexibilität um möglichst großen kreativen Freiraum für die Entwicklung von Prototypen in einem großen Anwendungsgebiet zu gewährleisten. Dabei spielen der Kostenfaktor, die Beschaffungsmöglichkeit und vor allem ein möglichst leichter Zugang wesentliche Rollen.

Mittlerweile ist auf dem Markt eine Vielzahl von Systemen erhältlich, die sich zu Forschungszwecken in der Robotik hervorragend eignen. Das Angebot reicht von schlichten programmierbaren Platinen, wie dem unscheinbaren Arduino (siehe Abbildung 1) der mit so gut wie jedem erhältlichen, elektronischen Bauteil kombinierbar und in einer Vielzahl von Sprachen programmierbar ist, bis zu fertigen Robotersystemen wie den humanoiden Roboter Nao (siehe Abbildung 3), der als Standardplattform für den RoboCup[1] dient, einem jährlichen Roboterfußballtunier, an dem sich Forschungsinstitute und Interessengruppen bei ihren Fortschritten messen können.

In der Mitte dieses Spektrums werden zahlreiche zumeist fertig konzipierte, sogenannte ‚educational robots' angeboten. Sie sind bereits mit einer Menge Sensoren und Aktoren ausgestattet, meist modular erweiterbar und mit höheren Programmiersprachen sehr leicht zu programmieren. Populäre Vertreter in diesem Segment sind z.B. der E-Puck (siehe Abbildung 2) oder der Scribbler[2].

Abbildung 1: Arduino Uno

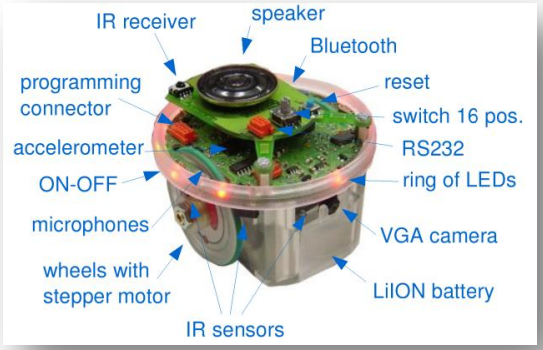

Abbildung 2: E-Puck

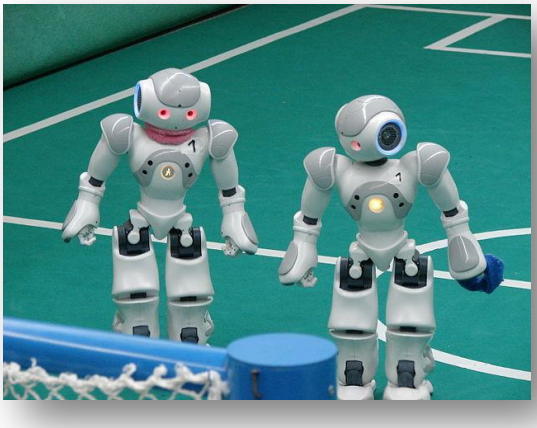

Abbildung 3: NAO Roboter

2.1.2 Lego MINDSTORMS NXT

Mit Lego, dem Bauklotz-Stecksystem, das der Kreativität kaum Grenzen setzt, wurde schon so manche kindliche Vision Realität. Mit Lego Technik, das pneumatische und mechanische Komponenten beinhaltet, schaffte es der Spielwarenhersteller schon 1977 eine ältere Zielgruppe an das System zu binden. Im Jahre 1986 verknüpfte Lego das Spielzeug erstmals mit Computertechnik, allerdings nur mit mäßigem Erfolg. Der nächste Versuch war 1998 mit der Lego MINDSTORMS RCX Serie, die sich als voller Erfolg zeigte. Herzstück war ein programmierbarerer Legostein mit LCD-Display an den zahlreiche Sensoren und Aktoren angeschlossen werden konnten. Es entwickelte sich schnell eine Fangemeinde rund um das modulare System, die bei weitem nicht nur aus Kindern bestand. Seit 2006 ist die verbesserte NXT-Serie erhältlich (siehe Abbildung 4), die zusätzliche Sensoren, eine Bluetooth-Schnittstelle, Lautsprecher und mit einen leistungsfähigeren Prozessor ausgestattet ist. Das System verfügt über einen 32Bit-Mikroprozessor, 256KB Flashspeicher und 64KB Ram. Das Starterset kostet um die 250€ und beinhaltet drei Servomotoren mit eingebauten Rotationssensor. Zudem wird neben zahlreichen Legosteinen, Kabeln und weiteres diverses Zubehör ein Tasten-, Ultraschall-, Farb- und Schallsensor mitgeliefert[3].

Da der Speicher des NXT flashbar ist, können beliebige Betriebssysteme zur Steuerung des NXT verwendet werden. Daher ist die Programmierung neben der von Lego nativen, leicht zugänglichen, grafikbasierten Programmierumgebung NXT-G auch in einer Vielzahl alternativer Sprachen möglich. Umgesetzt wurden neben dem viel genutzten, auf Java basierenden Betriebssystem LeJOS[4] auch welche, deren Syntax auf Assembler-, C, C++,C#, LUA oder Python basiert. Neben LeJOS ist auch das Echtzeitbetriebssystem nxtOSEK[5] erwähnenswert, das als einziges erlaubt nativ C- und C++-Code auszuführen.

Aufgrund dieser hohen Flexibilität, der einfachen Zugänglichkeit, dem Kosten- und Beschaffungsfaktor und vor allem dem modularen Prinzip findet das System in Hochschulen zu Lehr- und Forschungszwecken großer Beliebtheit.

So kommt auch bei dieser Projektarbeit ein Lego MINDSTORMS NXT zum Einsatz.

Abbildung 4: Ein Roboter realisiert mit Lego MINDSTORMS NXT

2.2 Roboternavigation

Es gibt unzählige Konzepte einem Roboter die Fähigkeit, autonom zu navigieren, zu verleihen. Hier wird darauf eingegangen, welche Voraussetzung seine Komponenten mit bringen müssen, um dies zu ermöglichen.

Roboter übernehmen in unserem Leben immer mehr Aufgaben. Dabei unterscheidet man zwischen stationären und mobilen Robotern, die entweder einem vorprogrammierten Ablauf folgen, ferngesteuert werden, oder aber auch vollkommen autonom agieren können. Damit sie dazu in der Lage sind benötigen sie Aktoren und meist auch Sensoren.

2.2.1 Aktoren und Sensoren

Besonders anspruchsvoll ist die Umsetzung autonomer mobiler Roboter, da sie sich in ihrer Umgebung orientieren und bewegen können müssen. Um diesen Ansprüchen gerecht zu werden benötigen sie Sensoren und Aktoren. Die Sensoren liefern dem Roboter Informationen über seinen eigenen Zustand (so genannte interne Sensoren) und dem der Umwelt in der er sich befindet (externe Sensoren). Ein interner Sensor gibt zum Beispiel Auskunft über die

verbleibenden Energie, oder der Umdrehungszahl eines Motors. Externe Sensoren versorgen den Roboter mit den Daten, die er benötigt um sich in seiner Umgebung zu Recht zu finden. Unter Aktoren versteht man alles, was dem Roboter dient seinen Zustand oder den der Umwelt zu verändern. Dabei dienen internen Aktoren der Zustandsänderung des Roboters, also beispielsweise Solarzellen, die seine Batterien laden und externe Aktoren der Interaktion mit der Umwelt, wie seine eigene Fortbewegung oder der von Objekten.

2.2.2 Navigation (Typen von Aktoren & Sensoren zu Navigation)

Es gibt eine Vielzahl von Möglichkeiten einen autonomen mobilen Roboter in seiner Umgebung navigieren zu lassen. Die externen Aktoren, die dabei zur Fortbewegung dienen, sind meist Räder, Ketten oder Beine aber können bei fliegenden Robotern auch Düsen, Rotoren oder Propeller sein.

Mindestens so zahlreich sind die unterschiedlichen externen Sensoren, die zur Orientierung

dienen können. Zur allgemeinen Orientierung im Raum kommen häufig Kompass oder Lagesensoren zum Einsatz. Um zu erkennen, ob der Roboter auf ein Hindernis stößt, wäre eine sehr einfache Lösung ihn mit einer Stoßstange auszustatten, die Kollisionen registriert. Passiert das wird eine programmierte Ausweichroutine durchgeführt. Will man Kollisionen möglichst vermeiden bieten sich Infrarot- oder Ultraschallsensoren an, die den Abstand zu Hindernissen messen können. Damit ist der Roboter in der Lage schon vor der Kollision ein Ausweichmanöver einzuleiten. Alternativ können auch Kameras zum Einsatz kommen, die dem Roboter räumliches Sehen, oder die Möglichkeit Muster zu erkennen, verleihen. Lichtsensoren liefern Farb- und/oder Helligkeitswerte zurück. So ist es beispielsweise möglich einer Lichtquelle oder Konturen zu folgen.

2.2.3 Linefollower

In dieser Projektarbeit kommt ein so genannter Linefollower zum Einsatz. Dies ist ein autonomer, mobiler Roboter, der in der Lage ist einer Linie auf dem Boden zu folgen. Er ist mit einem Farbsensor ausgestattet. Als externe Aktoren dienen 2 Räder, die unabhängig ansteuerbar sind. Der Farbsensor ist an der Front des Roboters auf den Boden gerichtet, damit er die Licht- und Farbwerte des Untergrundes messen kann. Der Roboter ist so programmiert, dass er der Kante einer schwarzen Linie auf weißem Untergrund folgen soll. Solange sich der Sensor mittig auf der Kante befindet, bewegt er sich vorwärts. Weichen die Messwerte ab, misst er also zu helle oder zu dunkle Töne, versucht der Roboter durch aufschaukelnde Links- und Rechtsbewegungen so lange seinen Kurs zu korrigieren, bis die Werte wieder im Toleranzbereich sind. [6]

2.3 Wegfindung

Es ist eine alltägliche Herausforderung einen Weg zwischen zwei Punkten zu finden. Wie muss ich in die Kurve fahren um möglichst wenig Geschwindigkeit zu verlieren, wie aber um möglichst wenig Zeit? Was ist der kürzeste Weg durch ein Labyrinth?

Wie man sieht, hängt es vor allem von dem Ziel ab, das man verfolgt, was den optimalen Weg ausmacht. Sehr häufig ist man an dem schnellsten oder kürzesten Weg interessiert. Heutige Navigationsgeräte bieten eine Reihe von Auswahlkriterien, nach denen man seine Route berechnen lassen kann. Dabei ist es auch von entscheidender Bedeutung, wie lange man warten muss, bis das Gerät mit den Berechnungen fertig ist, vor allem dann beispielsweise, wenn man von seiner Route abweicht. Dies hängt von der Leistungsfähigkeit der Hardware des Navigationsgeräts ab, aber noch viel mehr von der Effizienz der verwendeten Algorithmen.

In dieser Arbeit ist es ein Ziel die ideale Route, die der Linefollower bei einem Rundkurs, der aus Geraden und Kurven besteht, zu berechnen. Geht man davon aus, dass man den Roboter an der Innenlinie startet, ist die Anforderung an den gesuchten Algorithmus die kürzeste Route zwischen zwei Punkten, wobei der Startpunkt gleichzeitig der Zielpunkt ist.

2.3.1 Algorithmen zur Routenberechnung

Man unterscheidet bei Suchalgorithmen allgemein zwei Klassen: uninformierte und informierte. Uninformierte Suchalgorithmen vernachlässigen ein spezielles Suchproblem und können

deshalb sehr allgemein implementiert und universell eingesetzt werden. Allerdings sind die entstehenden Kosten, also die Suchzeit in der Regel sehr hoch, weshalb sie bei komplexeren Routenberechnungen mit vielen Wegknoten schnell unbrauchbar werden. Informierte Suchalgorithmen verwenden Strategien, meist Informationen oder Faustregeln, damit sie gleich in eine bestimmte, vermutete, wahrscheinlich günstigste ‚Richtung' suchen können, um den Prozess zu beschleunigen. Deshalb werden informierte Suchalgorithmen auch als heuristische Suchalgorithmen bezeichnet. Da der Start und das Ziel bei der Wegfindung in der Regel bekannt, also Informationen vorhanden sind, kann diese Klasse von Suchalgorithmen die Suchzeit erheblich verkürzen.

2.3.2 Der A*-Algorithmus

Die Vorgehensweise des Algorithmus basiert auf dem Shortest-Path-Algorithmus von Dijkstra. Laut Dijkstra werden ausgehend von einem Startknoten die Nachbarknoten verarbeitet. Ausgehend von den Nachbarknoten dann wiederum deren Nachbarknoten. Die Pfadkosten werden aufsummiert und es wird jeweils der Knoten zur Betrachtung der Nachbarn priorisiert, welcher die geringsten Pfadkosten besitzt. Somit ist auch sichergestellt, dass die erste gefundene Lösung auch eine optimale Lösung ist. Die Pfadkosten werden für jeden Knoten in Form einer Funktion abgebildet:

F(n) = g(n) mit g(n) = *Summe aller Pfadkosten vom Startknoten zu n über den kostengünstigsten Pfad*

A* erweitert diese Vorgehensweise um die Berücksichtigung einer Schätzfunktion h(n). Diese beschreibt für jeden Knoten n die geschätzte Entfernung zum Ziel:

f (n) = g(n) + h(n)

Als Summe der reellen Kosten vom Startknoten bis n und den geschätzten Kosten von n bis zum Ziel, beschreibt f(n) also eine Schätzung der gesamten Pfadkosten vom Startknoten bis zum Ziel über den Knoten n.

Wie auch bei Dijkstra priorisiert A* nun den Knoten mit dem geringsten f(n).

// mehr Detail

// Der oben informell beschriebene Ablauf wird wie folgt formalisiert. (-> Algorithmus)

2.4 Bildverarbeitung und Markererkennung

In diesem Kapitel werden Konzepte zur Bildverarbeitung und Markererkennung demonstriert, und Funktionsweise- und Umfang des ARToolKit vorgestellt.

// Motivation (Kurserkennung, Roboterlokalisierung)

Die heutigen Möglichkeiten der Bildverarbeitung bieten sich hervorragend an, um mit geringem Aufwand ohne Einsatz von Messinstrumenten das Geschehen in Videobildern zu verarbeiten und zu analysieren.

//evtl noch alternativen, markerloses tracking, infrarotmarker

Vor allem die Erkennung markanter Konturen und Muster funktioniert mittlerweile sehr zuverlässig. So lässt sich der Kurs eines Linefollowers, der von einer schwarzen Linie auf weißem Untergrund repräsentiert wird, leicht von einem Computer in einem Bild identifizieren. Genauso verhält es sich mit sogenannten Markern, die vor allem in der Augmented Reality Anwendung finden. Ein Marker ist ein kontrastreiches Muster, das so konzipiert ist seine Lage

im Raum eindeutig zu identifizieren (siehe Abbildung 5).

Abbildung 5: Beispiel für einen Marker

2.4.1 Frameworks zur Markererkennung

Es gibt eine Vielzahl von Frameworks, die auf dem Prinzip der Markererkennung basieren. Sehr populär ist das ARToolkit[7], das als Grundlage vieler, meist kommerzieller Weiterentwicklungen Verwendung fand. Dazu zählen zum Beispiel das ARToolkitPlus[8], ARToolkitPro[9], OSGART[10] und der für Mobilgeräte spezialisierte Studierstube Tracker[11] die sich durch einen erweiterten Funktionsumfang und einer optimierten Markererkennung auszeichnen.

//ausbaubar

2.4.2 ARToolKit

Um eine Analyse der Videodaten zu ermöglichen, wurde das Software-Framework ARToolkit verwendet. Es bietet neben dem Zugriff auf Videodaten eine robuste und einfache Markererkennung, die zum automatischen Lokalisieren des Roboters und des Kurses im Kamerabild verwendet wird.

//Gründe für die Verwendung von ART

Das ARToolkit ist ein sehr populäres, zuverlässiges und kostenloses Software-Framework zur Realisierung von Augmented Reality Anwendungen. Hier wird es allerdings zur automatischen Lokalisierung des Roboters und des Kurses im Kamerabild verwendet.

Die Software erkennt in den Bildern von Videoquellen vorprogrammierte, quadratische Marker und liefert die Markeridentifikationsnummer und die Positionen seiner vier Eckpunkte zurück. Aus diesen Informationen wird die Position der Kamera berechnet, aus dieser sich dann, unter Hilfe von OpenGL, virtuelle Objekte relativ zur Markerposition im Videobild darstellen lassen (siehe Abbildung 6). In dieser Arbeit wird das Zentrum der vier Eckpunkte als Positionspunkt des Roboters verwendet. [12]

//evtl noch paper mit effizenzvergleich der frameworks

Abbildung 6: Ein auf ein Marker projiziertes 3D-Modell

3 RoboTracker

Dieses Kapitel bietet einen detaillierten Einblick in den Funktionsumfang und der Architektur des, auf dem ARToolKit basierenden Frameworks. Außerdem werden die funktionsgebenden Implementierungen näher beleuchtet.

3.1 Motivation

Um die Qualität einer Linefollower-Implementierung mess- und vergleichbar zu machen, wurde im Rahmen dieser Projektarbeit einerseits eine Metrik zur Berechnung der Qualität eines Linefollowers erstellt; andererseits eine Anwendung zur automatischen Berechnung dieser Kenngröße implementiert. Das Framework ermöglicht es, diese Metriken an die Ansprüche des Experiments beliebig anzupassen. Voraussetzung ist lediglich der Sichtkontakt vom Objektiv der Kamera zum Marker.

In dieser Arbeit wird ein bestehender Roboter mit einem Marker versehen, der von einer Kamera aufgenommen wird und so mittels der Anwendung roboTracker eine automatische Analyse des Roboterverhaltens ermöglicht. Diese Analyse soll die Genauigkeit des zum Einsatz kommenden Linefollowers festhalten und nachvollziehbar machen.

3.2 Funktionsumfang

Das Programm roboTracker stellt verschiedene Funktionalitäten zur Analyse von Linefollower zur Verfügung. Dazu gehört die Einteilung einer experimentellen Umgebung in verschiedene Umgebungstypen anhand der Bilddaten, des weiteren die Berechnung der Ideallinie durch einen Kurs und die automatische Erkennung eines Markers zur weiteren Analyse des Verhaltens eines markierten Roboters; sowie Funktionalität zur weiteren Unterstützung dieser Analyse.

Des weiteren ist es möglich verschiedenste USB- oder Firewirekameras in Echtzeit in beliebigen Auflösungen und Bildwiederholraten auszuwerten. Desweiteren können auch vorher aufgezeichnete Experimente in einer Vielzahl von Videoformaten ausgewertet werden.

3.2.1 Umgebungsanalyse

Allgemein erfolgt die Analyse des Kurses nach folgendem Schema: Jeder Bildpunkt wird anhand seiner Farbeigenschaften einem bestimmten Umgebungstyp zugeordnet. Seien beispielsweise verschiedene Umgebungstypen in einem Kurs abgebildet (z.B. Strecke, Nicht-Strecke, Start- und Zielzone), so kann nach dem hier beschrieben Vorgehen jeder Bildpunkt eindeutig einer dieser Klassen zugeordnet werden.

Umgebungsanalyse: Farbwerte Bildpunkt -> Umgebungstyp.

Zunächst wird anhand eines Videoframes der im Experiment festgelegte Kurs analysiert. Der vom ARToolKit gelieferte Bilddatenvektor enthält für jeden Bildpunkt die Farbwerte im Format ABGR (Alpha-, Blau-, Grün- und Rotwert). Aus diesen Werten wird nach folgender Formel ein annähernder Helligkeitswert berechnet:

$$\frac{Blauanteil + Grünanteil + Rotanteil}{3}$$

Wenn der so berechnete Helligkeitswert einen bestimmten Grenzwert unterschreitet, wird der Bildpunkt der Strecke zugeordnet, im anderen Fall gehört der Punkt nicht zum Kurs.

Zur weiteren Unterteilung des Videobildes in unterschiedliche Farbbereiche kann optional auch geprüft werden, ob der Farbwert eines Bildpunkts in der Nähe eines Zielfarbwerts angesiedelt ist. Dazu wird jeder Farbanteil des Bildpunkts mit dem entsprechenden Anteil der Zielfarbe verglichen. Ist die Differenz des Bildpunkts zur Zielfarbe klein genug, so wird der Punkt dem entsprechenden Umgebungstyp zugeordnet.

Die so berechneten Umgebungstypen werden für jeden Pixel in einem zweidimensionalen Vektor hinterlegt.

3.2.2 Berechnung der Ideallinie und Start-bzw. Zielpunkt

Um die Berechnung der Ideallinie durch einen Kurs zu ermöglichen, müssen zunächst Start- und Zielpunkt der vom Roboter zu verfolgenden Strecke festgelegt werden. Es werden zwei Fälle unterschieden:

- Geschlossene Strecke (Rundkurs)
- Offene Strecke

Zur Vereinfachung wird im Folgenden von einer Ideallinie ausgegangen, deren Start- und Zielpunkt an der Innenkante der geschlossenen Strecke platziert sind.

Im Falle einer geschlossenen Strecke werden Start- und Zielpunkt automatisch berechnet. Damit der verwendete A*-Algorithmus zur Berechnung der Ideallinie sinnvoll angewandt werden kann, muss der Rundkurs an einer Stelle unterbrochen werden, da Start- und Zielpunkt sonst direkt nebeneinander liegen und die Ideallinie (als kürzeste Strecke zwischen zwei Punkten) somit nicht sinnvoll verwendet werden könnte.

Zu diesem Zweck wird der zweidimensionale Umgebungstypvektor in seiner Mitte vertikal von oben nach unten durchlaufen. Der erste so gefunden Kursbildpunkt wird aus dem Kurs entfernt, ebenso alle direkt darauf folgenden Kursbildpunkte; dies geschieht so lange, bis ein Punkt gefunden wird, der nicht zum Kurs gehört (siehe Abbildung 7).

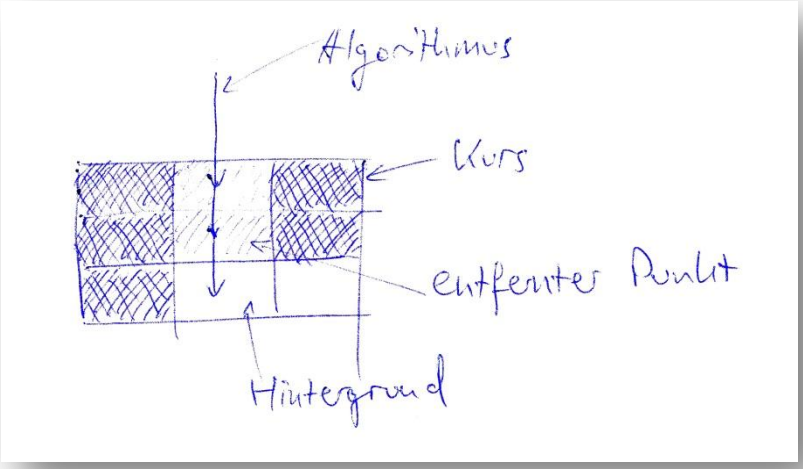

Abbildung 7: Rundkurstrennung

Dann werden zur Berechnung von Start- und Zielpunkt jeweils die beiden Bildpunkte rechts und links vom zuletzt besuchten Punkt betrachtet. Sind diese Teil des Kurses, so werden sie als Start- bzw. Zielpunkt festgelegt. Andernfalls wird jeweils der Bildpunkt darüber auf die gleiche Weise behandelt. Dies führt in jedem Fall spätestens beim dritten Analyseschritt zu einem validen Ergebnis (siehe Abbildung 8).

Sollte der Kurs an der Außenkante betreten und wieder verlassen werden, liefert er zwar Start

und Zielpunkt, diese sind allerdings an der Außenkante platziert.

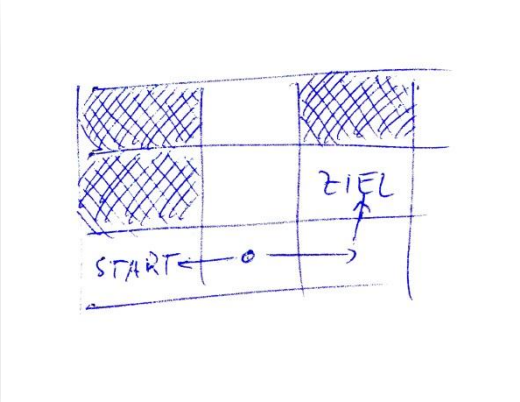

Abbildung 8: Start- und Zielpunktberechnung

Wenn der Kurs an keiner Stelle durchschnitten wird, muss ein erneuter Durchlauf des Umgebungstypvektors an anderer Stelle erfolgen.

Im Falle eines offenen Kurses werden Start- und Zielpunkt manuell mit Hilfe der Maus gesetzt.

Nachdem Start- und Zielpunkt der Strecke bestimmt wurden, wird der A*-Algorithmus zur Berechnung der Ideallinie angewendet.

3.2.3 Aufzeichnung, Analyse und Auswertung der Daten

Um verschiedene Linefollower-Implementierung vergleichbar zu machen, ist zunächst die Definition praktisch anwendbarer Metriken notwendig:

- Durchschnittliche Abweichung des Roboters von der Ideallinie
- Maximale Abweichung des Roboters von der Ideallinie
- Durchschnittlich benötigte Zeit pro Runde

// Berechnung der Abweichung

Zur Bestimmung der jeweils aktuellen Position des Roboters wird das ARToolKit verwendet, das den Mittelpunkt eines am Roboter angebrachten Markers als Ergebnis liefert.

Die aktuelle Abweichung berechnet sich aus der geringsten euklidischen Distanz aller Punkte des Ideallinienvektors zur Position des Roboters.

// Screenshot

Alle so erhaltenen Daten werden in einem Vektor gespeichert und am Ende des Experiments zur Berechnung der oben angegebenen Metriken verwendet. Die einzelnen Metriken berechnen sich dabei wie folgt:

$$Abweichung\ \emptyset = \frac{\sum Abweichung\ zum\ Zeitpunkt\ t}{Anzahl\ Zeitpunkte}$$

$$Abweichung\ \max = \max\left(Abweichungsvektor\right)$$

$$\frac{Zeit}{Runde}\emptyset = \frac{Zeit\ gesamt}{Anzahl\ Runden}$$

// Speichern, Laden, Logfunktion

RoboTracker ermöglicht das Speichern und Laden von Umgebungsvektoren und Experimentalergebnissen.

3.3 Architektur

Das Framework ist in Pakete gegliedert, um die Wartbarkeit zu gewährleisten und um eventuelle Erweiterungen einfach zu ermöglichen. Dabei wird den Prinzipien größtmöglicher Kohäsion (voneinander abhängige Komponenten werden zusammengefasst) und loser Kopplung (möglichst wenig Abhängigkeiten der Komponenten untereinander) Rechnung getragen.

Um das Framework soweit möglich von technischen Detailimplementierungen (z.B. verwendete Kamera, Input-Management, etc.) unabhängig zu halten, besitzt jedes Paket eine Fassadenklasse, die die Spezifika der jeweiligen Implementierung für das Framework transparent hält. Um also beispielsweise Funktionalität der Implementierung des *Video*-Pakets zu verwenden, greift der Framework-Kern lediglich auf die *Video*-Fassadenklasse zu, die etwaige Aufrufe dann an die internen Paket-Klassen des *Video*-Pakets weiterleitet.

Das Framework besteht aus fünf Paketen: Ein Haupt-Paket (*Main*), eines zum Umgang mit Video-Daten und der Markeranalyse (*Video*), ein Paket zur Berechnung relevanter Daten für das Experiment (*Analyse*), eines für Visualisierung und Eingabebehandlung (*I/O*), sowie einem Paket zur abstrakten Repräsentation des Experiments (*Model*).

Dabei greift das *Main*-Paket auf die drei funktionalen Pakete *Video*, *Analyse* und *I/O* zu; diese funktionalen Pakete wiederum lesen und schreiben das repräsentierende *Model*-Paket. (siehe Abbildung 9)

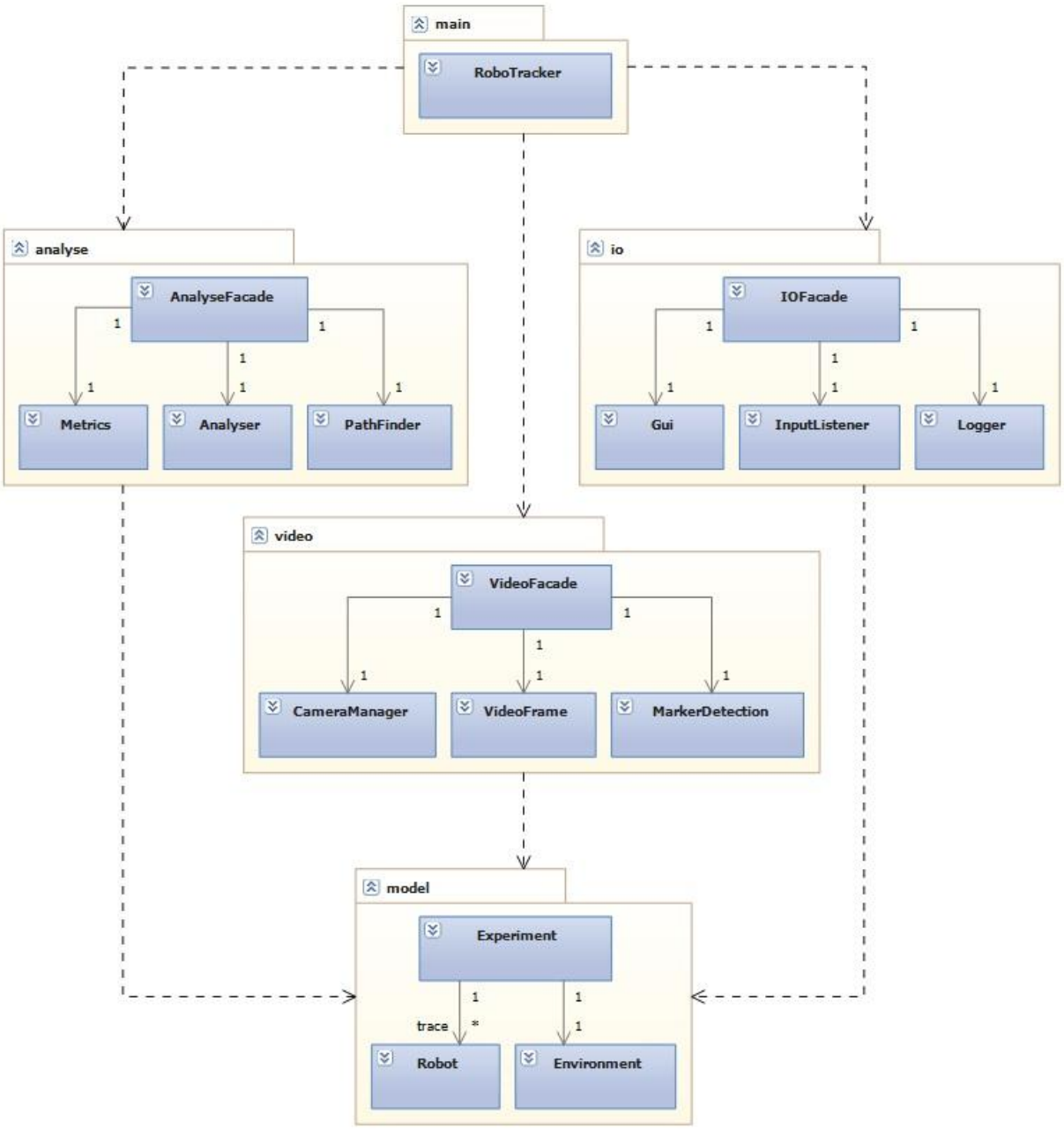

Abbildung 9: Klassendiagramm des RoboTracker

3.3.1 Pakete

Die verschiedenen Pakete übernehmen dabei wie oben beschrieben unterschiedliche Aufgaben, die im folgendem genauer dargestellt werden.

- Main

Main ist das Haupt-Paket des Frameworks. Es enthält das Hauptprogramm zur Linefollower-Analyse. Dieses verwendet die funktionalen Pakete *Video*, *Analyse* und *I/O*, um Ergebnisse zu berechnen und darzustellen. Das *Main*-Paket besitzt keine Fassadenklasse, da es von den anderen Paketen nicht aufgerufen wird.

- Model

Dieses Paket enthält Klassen zur abstrakten Repräsentation des Linefollower-Experiments. So werden der Kurs und andere Merkmale der Experimentalumgebung hier als Datentypen repräsentiert. Dazu zählen die Ideallinie im Kurs sowie der Roboter (Position, Abweichung von der Ideallinie, etc.) und der Verlauf des Experiments (aktuelle Runde, Verlauf der Roboterposition, etc.).

Das Model wird von drei Paketen verwendet:

- *Video*, um aus den Bilddaten die Umgebung zu berechnen
- *Analyse*, um den aktuellen Status und den Verlauf des Experiments zu erfassen
- *I/O*, um das Experiment und die Ergebnisse der Analyse zu visualisieren und um evtl. Eingabeereignisse zu verarbeiten, die das Modell betreffen.

- Video

Dieses Paket bietet Funktionalität um Frames des Input-Videostreams zugreifbar zu machen, sowie verschiedene Möglichkeiten zur Analyse dieser Frames. So kann z.B. aus den Bilddaten der Kamera das abstrakte Framework-Modell der Umgebung erzeugt werden, um den Kurs, freie Flächen und andere Umgebungstypen zu lokalisieren. Auf Basis dieser Umgebungsdaten kann dann auch die Ideallinie durch den Kurs berechnet werden. Außerdem wird hier das Markertracking (also die Lokalisierung des Roboters) vorgenommen.

Um lose Kopplung zu erreichen, wird das *Video*-Paket ausschließlich vom *Main*-Paket verwendet.

Die Fassadenklasse bietet folgende Funktion an:

- *computeRobotPosition*:
  Die Position des Roboters wird anhand des daran angebrachten Markers vom *ARToolkit* berechnet (siehe Kapitel 2.4.2). Das Ergebnis wird im *Model* gespeichert.

- Analyse

Das *Analyse*-Paket enthält Klassen zur Analyse des Umgebungsmodells sowie zur Berechnung von Experimentalergebnissen. Dazu gehören die Berechnung der Ideallinie sowie die Berechnung des Abstands zwischen aktueller und idealer Roboterposition in jedem Frame.

Um lose Kopplung zu erreichen, wird das *Analyse*-Paket ausschließlich vom *Main*-Paket verwendet.

Die Fassadenklasse bietet folgende Funktionen an:

- *analyse*:
  Hier wird die Distanz des Roboters zur momentanen Idealposition berechnet (siehe Kapitel 3.2.3). Dieser Abstand wird im Modell gespeichert.
- *computeMetrics*:
  Hier werden die im Kapitel 3.2.3 beschriebenen Metriken berechnet.
- *computeIdealPath*:
  Diese Funktion berechnet die Ideallinie in einem Rundkurs mit Hilfe der Implementierung aus der Klasse Pathfinder des *Analyze*-Package. In der Beispielimplementierung wird der A*-Algorithmus verwendet (siehe Kapitel 2.3.2).

- I/O

Das *I/O*-Paket übernimmt einerseits die Visualisierung des Experiments und der vom Framework errechneten Ergebnisse (z.B. Tracking der Roboterposition, Visualisierung des

abstrakten Umgebungsmodells, Abstand zur Ideallinie). Andererseits ist es für die Verarbeitung etwaiger Eingabeereignisse zuständig.

Um lose Kopplung zu erreichen, wird das *I/O*-Paket ausschließlich vom *Main*-Paket verwendet.

Die Fassadenklasse bietet folgende Funktionen an:

- *logState*:
  Der aktuelle Zustand des Experiments wird in eine Textdatei geschrieben.
- *renderState*:
  Die gesammelten Daten des Experiments.werden visualisiert, sofern gewünscht.
- *getInputEvents*:
  Diese Funktion liefert eine Liste aller zu verarbeitenden Benutzereingaben.

3.3.2 Programmsequenz

Abbildung 10 zeigt schematisch den Ablauf des Mainloops des RoboTracker-Programms.

Bei Start des Programms werden GUI, Inputhandling und Logging initialisiert. Dann wird anhand eines Kameraframes die abstrakte Repräsentation der Experimentalumgebung erstellt. Auf dieser Grundlage wird die Ideallinie durch den Experimentalkurs berechnet. Danach wird die Hauptschleife des Programms gestartet.

In der Hauptschleife (siehe Abbildung 10) werden zunächst die Eingabeereignisse vom *I/O*-Paket abgefragt und dann verarbeitet. Anschließend wird die Position des Roboters im Kameraframe ermittelt und das Modell entsprechend aktualisiert.

Dann berechnet das *Analyse*-Paket anhand der Position des Roboters seine Abweichung zur Ideallinie (siehe Kapitel 3.2.3). Diese wird ebenfalls im *Model* hinterlegt. Außerdem werden die in Kapitel 3.2.3 definierten Metriken berechnet.

Zuletzt wird der aktuelle Modellzustand in einer Textdatei geloggt und in der GUI visualisiert.

Die Hauptschleife endet, nachdem der Roboter die im Experiment festgelegte Rundenanzahl absolviert hat.

Nachdem das Experiment abgeschlossen wurde, werden die berechneten Metriken in einer separaten Textdatei geloggt.

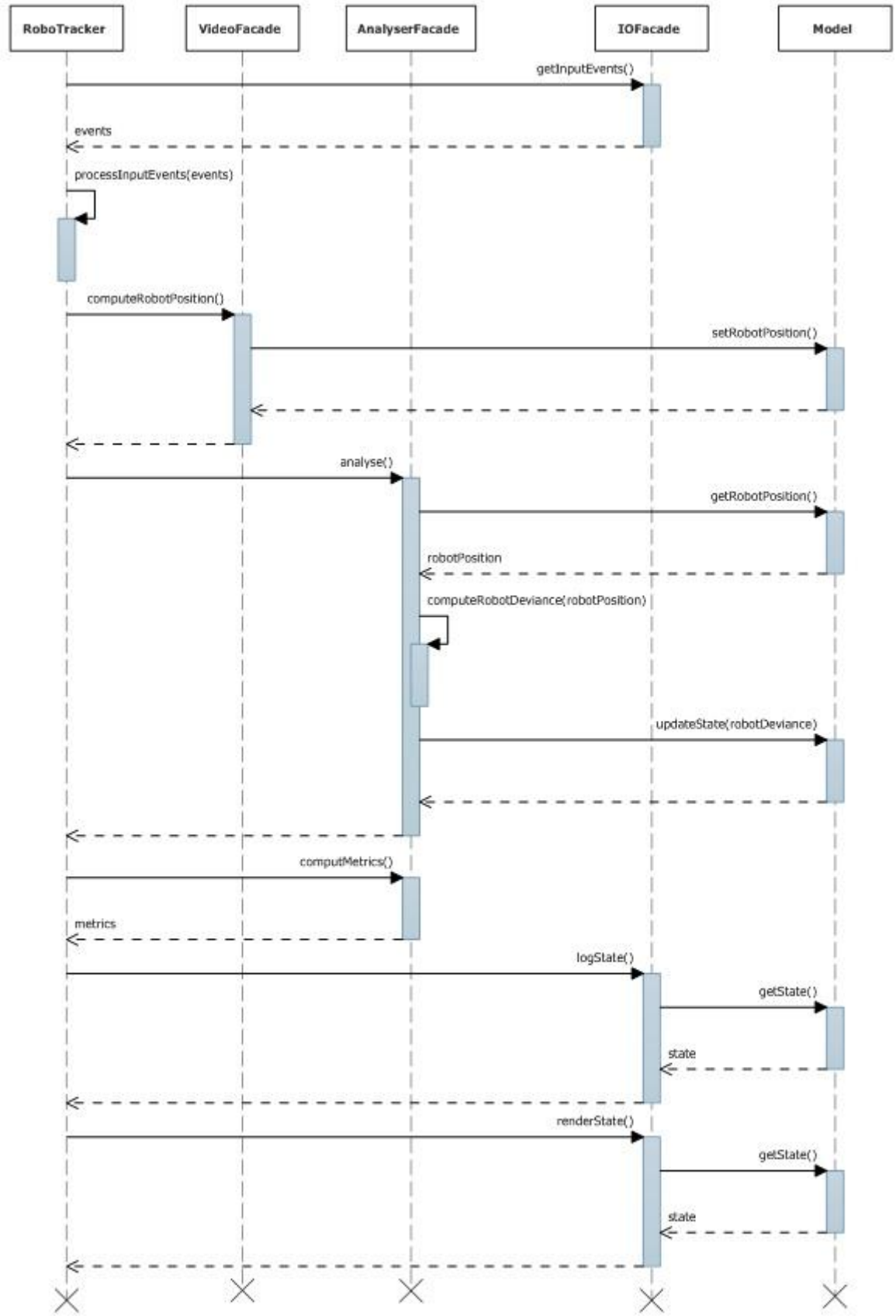

Abbildung 10: Hauptschleife des RoboTracker

4 Experiment

In diesem Kapitel wird der praktische Einsatz des RoboTracker, anhand eines Linefollower-Roboters demonstriert. Es werden zwei Versuche auf zwei unterschiedlich gestalteten Kursen durchgeführt. Dabei wird im ersten Versuch in zwei Durchläufen verglichen, wie sich der dem linken Rand folgende Linefollower, über mehrere Runden einmal im und einmal gegen den Uhrzeigersinn fahrend, auf dem einfachen Testparcours schlägt. Dieser Vorgang wird mit einem komplexeren Kurs wiederholt. Bei diesem Versuch werden die Videobilder nicht Echtzeit analysiert, sondern erst nachträglich in das Programm geladen und ausgewertet.

Die vom RoboTracker ermittelten Daten werden anschließend ausgewertet, um fest zu stellen, welche Varianten der Kurse dem Roboterlayout und seiner Implementierung mehr oder weniger Probleme bereiten.

4.1 Aufbau

Der Versuchsaufbau besteht aus zwei schwarz auf weißen Papier gedruckten Rundkursen auf denen sich der Roboter orientieren und bewegen soll. Darüber ist eine Kamera angebracht, die das Geschehen zur späteren Analyse aufzeichnet. Der Kurs ist auf einem ein Meter breiten und zwei Meter langen Korkbrett befestigt. Um eine ausreichende Beleuchtung zu gewährleisten befinden sich an den Längsseiten des Aufbaus zwei 180cm lange Leuchtstoffröhren, die auf niedriger Höhe angebracht sind. Dadurch soll die Funktion des Lichtsensors unterstützt und der Marker stets ausreichend beleuchtet werden, um die spätere Identifikation über das *ARToolkit* zu gewährleisten. (siehe Abbildung 11)

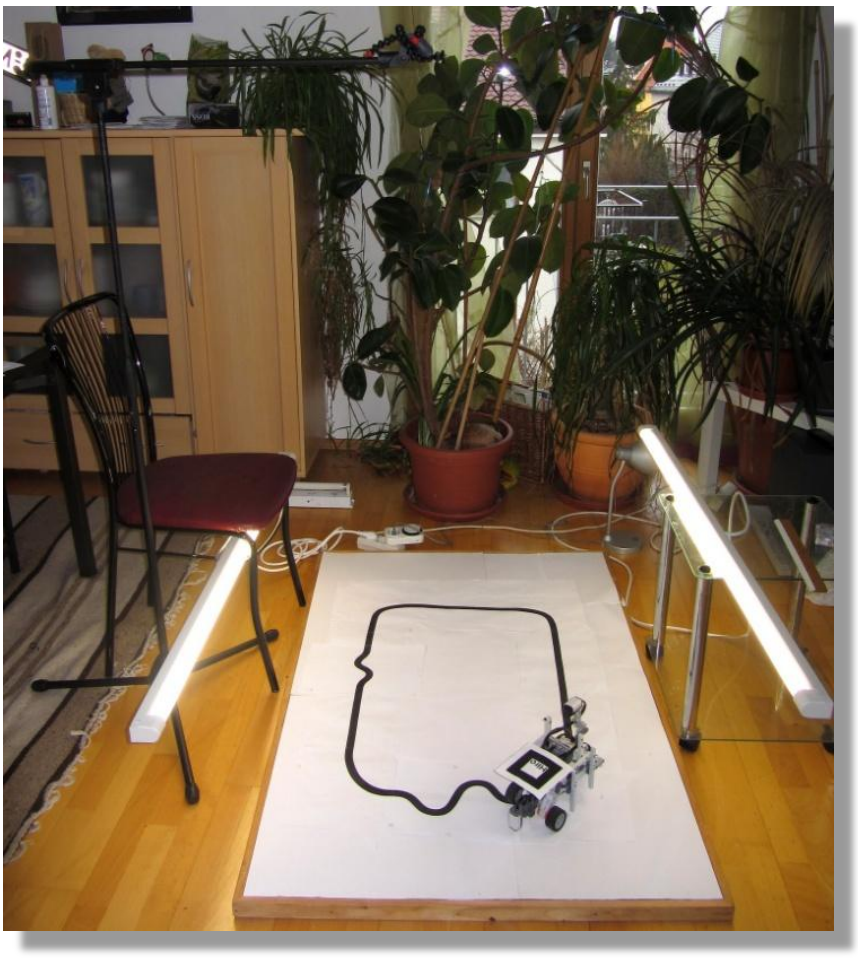

Abbildung 11: Der Versuchsaufbau ohne Kamera

Außerdem wird dadurch evtl. störender Schattenwurf des Roboters minimiert. An einem metallischen Mikrofonstativ ist die Kamera mit Hilfe eines magnetischen Kompaktkamerastativs befestigt. Dies ermöglicht eine relativ exakte und bequeme Ausrichtung über dem Kurs. Dabei ist die Kamera ca. zwei Meter zentral, über dem Aufbau platziert und senkrecht auf den Kurs gerichtet, um die perspektivische Verzeichnung möglichst gering zu halten, da diese zu Problemen mit der exakten Markererkennung und damit der Präzession, der gesammelten Daten führen könnte.

4.1.1 Kurse

// TODO: Erkennungsfaktor möglichst hoch -> Quelle von Rechner daheim -> most confidential… <- noch rein

Das Experiment wird auf zwei unterschiedlichen Rundkursen durchgeführt.

Der erste einfache Rundkurs besteht aus zwei weiten und zwei schärferen Kurven, die mit zwei kürzeren und zwei längeren Geraden verbunden sind (siehe Abbildung 12).

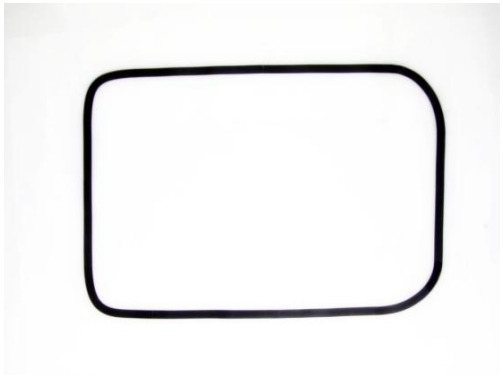

Abbildung 12 – einfacher Kurs

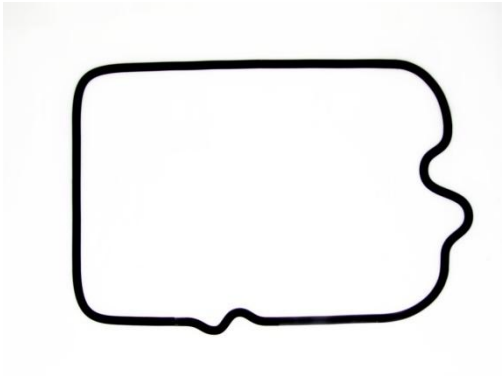

Abbildung 13 - komplexerer Kurs

Der zweite komplexere Kurs ist eine Erweiterung des ersten Kurses. In der Mitte einer langen Geraden wurde eine leichte rechts-links-rechts Schikane ergänzt. Eine kurze Gerade wurde mit einer engkurvigeren Version der ersten Schikane ersetzt (siehe Abbildung 123)

Die einzelnen Elemente der Kurse wurden auf DIN-A4 Blättern gedruckt, passend zusammengeklebt und mit Reißnägeln auf einem Korkbrett fixiert, um mögliches Verrutschen, verursacht durch die Traktionskraft der Antriebsräder des Roboters, zu verhindern.

4.1.2 Roboter

Der Versuchsroboter ist ein mit Lego MINDSTORMS realisierter Linefollower. Er bewegt sich auf drei Rädern. Dabei besitzt er vorne als Antriebsaktoren zwei große, starr vorwärts ausgerichtete Antriebsräder, die mit einem weichen Gummiprofil beschichtet sind und sich unabhängig voneinander in verschiedenen Geschwindigkeitsstufen vorwärts und rückwärts bewegen können. Das dritte, passive Rad ist zentral am Heck des Roboters montiert und nicht starr befestigt, sondern lässt sich um 360° über seine Hochachse rotieren. Dadurch gewährt es dem Roboter die nötige Stabilität um nicht umzufallen und lenkt in jede eingesteuerte Richtung mit, ohne quer zu stehen und damit die Drehung zu blockieren.

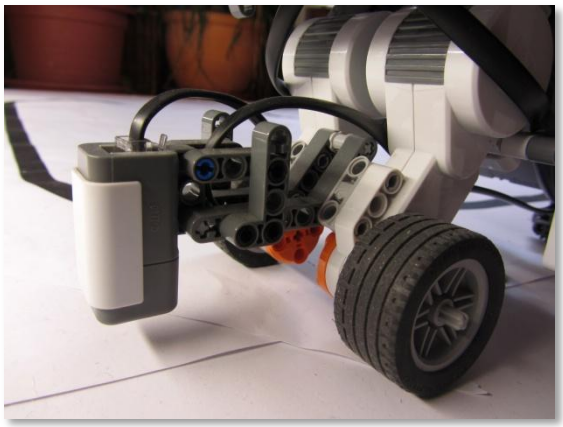

Abbildung 14: Der Farbsensor

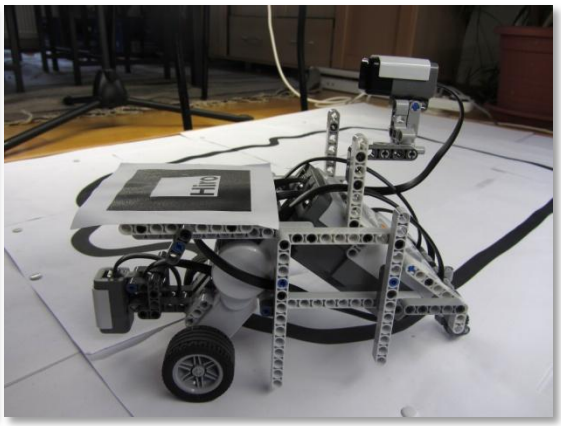

Abbildung 15: Der Linefollower von der Seite

Der Roboter besitzt einen Farbsensor, der mittig an der Front angebracht ist. Er ist senkrecht auf den Boden gerichtet und befindet sich auf ca. 1cm Höhe (siehe Abbildung 14). Er dient dem Linefollower zur Unterscheidung der Farbwerte zwischen schwarz und weiß und beleuchtet den Untergrund, den er analysiert, mit einer blauen LED.

Die zentrale Steuereinheit befindet sich im Zentrum des Roboters und ist vorne nach oben geneigt installiert. Das ermöglicht einen bequemen Zugriff auf die Bedienelemente und bietet viel Raum für die Verkabelung der Komponenten

Das Zentrum des quadratischen Markers liegt auf der Drehachse des Roboters. Er ist aus Gründen der perspektivischen Verzeichnung so tief wie möglich angebracht und parallel zum Boden ausgerichtet. (siehe Abbildung 14).

4.1.3 Kamera

Zur Aufzeichnung der Videos ist eine Canon SX230 HS Fotokompaktkamera zum Einsatz gekommen [13]. Sie ist mit einem 16 Megapixel CMOS-Sensor und einem Zoomobjektiv mit einer Brennweite von 28 bis 392 Millimeter und einer Lichtstärke von F3.1 bis F8.0 ausgestattet. Die Kamera besitzt einen sehr guten Videomodus, der Aufnahmen bis zu 1920x1080 Bildpunkten mit 24 Bildern pro Sekunde ermöglicht. Zudem verfügt sie über ein manuelles Programm, das es erlaubt den Weißabgleich, den Fokusierungspunkt und die Brennweite während der Videoaufzeichnung fix festzulegen.

Die Versuche wurden mit 29,97 Bildern pro Sekunde bei einer Auflösung von 640x480 Bildpunkten im H.264-MPEG 4-AVC-Codec aufgezeichnet. Der Focus war manuell auf den Boden fixiert. Das Objektiv war auf Weitwinkel (28mm) eingestellt. Die Kamera wurde in zwei Metern Höhe, zentral über dem Kurs an einem Stativ montiert und frontal auf ihn gerichtet. Es wurde eine Kleinbildbrennweite von 55mm bei einer Blende von F4.0 gewählt. Bei dieser Brennweite ist die Verzeichnung der Optik des Objektivs besonders gering.

4.2 Durchführung

Vor jeder Durchführung eines Versuchsdurchlaufs muss der Farbsensor des Linefollowers zunächst auf die Helligkeitswerte des Kurses kalibriert werden. Dabei wird zunächst der Weißwert, der neben dem Kurs liegt, und anschließend der Schwarzwert der zu folgenden Linie gemessen.

Als nächstes wird die Kamera korrekt eingestellt und so ausgerichtet, dass ausschließlich der Kurs im Bildausschnitt zu sehen ist. Nun wird die Videoaufzeichnung gestartet.

Zunächst wird für einige Sekunden der Kurs gefilmt, damit der RoboTracker in der Lage ist den Kurs korrekt zu erfassen. Nun wird der Roboter auf die Linie des Kurses gesetzt. Hat der Roboter fünf Runden absolviert, wird die Datenaufzeichnung gestoppt und ausgewertet.

Dieses Vorgehen ist bei allen vier Versuchskonfigurationen identisch.

Zur anschließenden Auswertung der Videodaten werden diese von der Kamera auf den Computer übertragen, in das H.264-AVI-Format konvertiert und im RoboTracker einzeln zur Analyse geladen.

Sobald das Programm gestartet ist, wird der Kurs erfasst, seine Ideallinie berechnet und die Start-bzw. Zielpunkte gesetzt. Überquert der Roboter das erste Mal den Startpunkt beginnt die erste Runde und die Aufzeichnung der Daten. Diese wird wiederum nach der vorher festgelegten Anzahl von Runden beendet und die vorher festgelegten Metriken berechnet.

4.3 Ergebnis

Die ersten beiden Versuchsdurchläufe, wurden auf dem einfachen Kurs (siehe Kapitel 4.1.1) durchgeführt. Dabei fuhr der Linefollower den identischen Kurs einmal im und einmal gegen den Uhrzeigersinn über 5 Runden ab.

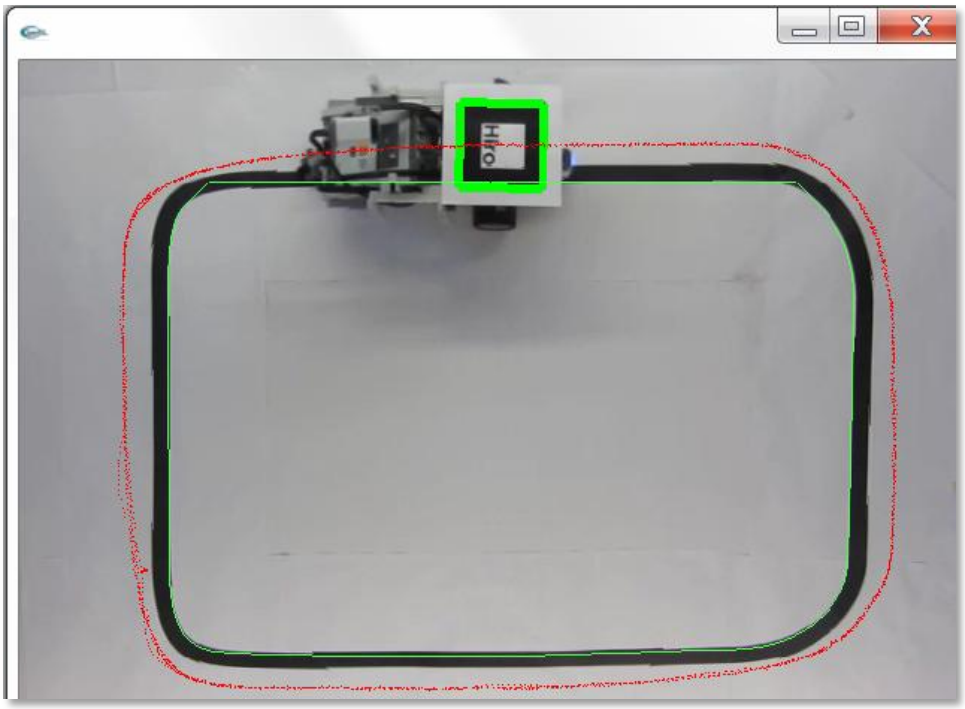

Abbildung 16 - aufgezeichneter Kurs des Linefollowers nach fünf Runden im einfachen Kurs im Uhrzeigersinn abgefahren

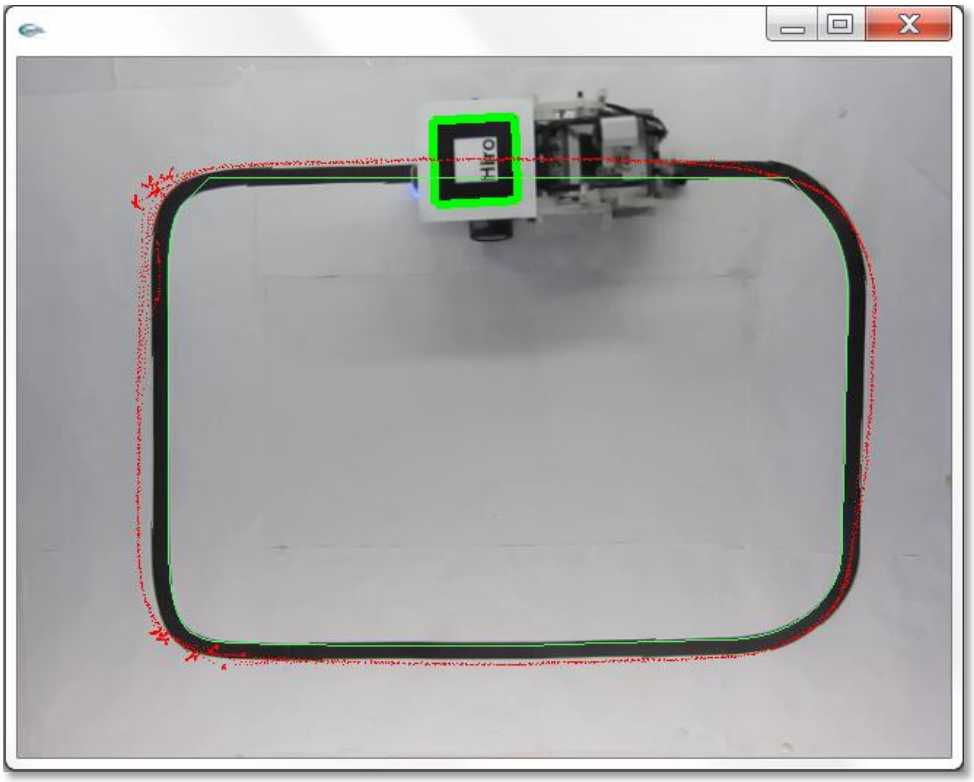

Abbildung 17 - aufgezeichneter Kurs des Linefollowers nach fünf Runden im einfachen Kurs gegen den Uhrzeigersinn abgefahren

Dabei kam es zu folgenden Ergebnissen:

Einfacher Kurs	mittlere Abweichung	max. Abweichung	mittlere Rundenzeit
Uhrzeigersinn	3.23881 cm	4.91656 cm	**16.8968** sec
vs. Uhrzeigersinn	**1.97017** cm	4.44225 cm	21.3794 sec

Einfacher Kurs	Runde 1	Runde 2	Runde 3	Runde 4	Runde 5	Gesamtzeit
Uhrzeigersinn (in sec)	16.165	15.999	**18.053**	16.917	17.350	**84.484**
vs. Uhrzeigersinn (in sec)	20.474	21.504	21.047	22.223	21.649	106.897

Abbildung 18: tabellarische Auswertung der Metriken im einfachen Kurs

Die Auswertung der Metriken können der Abbildung 18 entnommen werden.

Der Linefollower brauchte für die fünf Runden im einfachen Kurs, gefahren im Uhrzeigersinn 84.484 Sekunden, und folgte dabei der Außenlinie. Dabei betrug die mittlere Abweichung 3.23882 Zentimeter und die maximale Abweichung 4.91656 Zentimeter. Er kam nur einmal in der dritten Runde leicht von der Linie ab und musste seine Ausrichtung korrigieren. Ansonsten konnte er den Kurs, ohne stoppen zu müssen, absolvieren. Dies kann man visuell auch dem sehr gleichmäßigen Verlauf der Messpunkte entnehmen (siehe Abbildung 16).

Im Gegensatz dazu betrug die Gesamtzeit für die fünf Runden, auf dem gleichen Kurs gegen den Uhrzeigersinn gefahren 21.3794 Sekunden, also deutlich länger als im ersten Versuchsdurchlauf. Der Roboter fuhr die Innenlinie ab. Die maximale Abweichung betrug

4.44225 Zentimeter bei einer mittleren Abweichung von 1.97017 Zentimeter. Während der Roboter die weiteren Kurven ohne Kursabweichung und einem daraus resultierenden Korrekturmanöver abfahren konnte, kam er bei den beiden engeren Kurven in jeder Runde stets von der Linie ab und musste diese daraufhin durch mehrere immer weitere Drehbewegungen wieder finden. Die Stellen an denen dies notwendig wurde, erkennt man gut an den größeren Punktehaufen an den beiden engen Kurven auf der linken Seite der Abbildung 17.

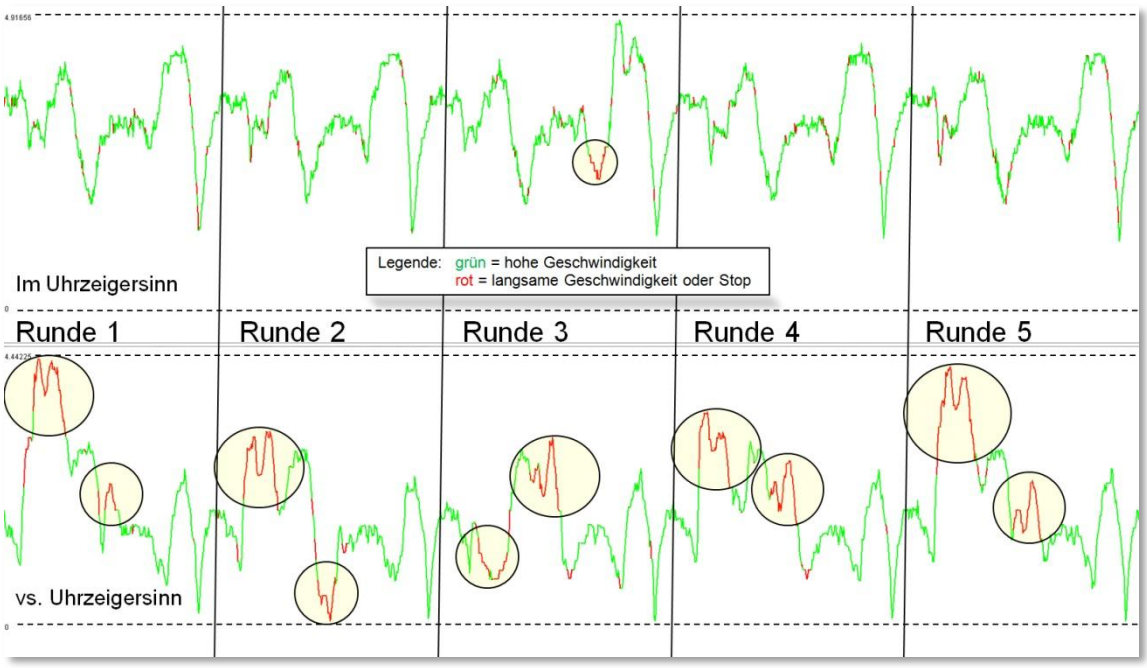

Abbildung 19: graphische Auswertung des einfachen Kurses

Abbildung 19 zeigt eine mögliche graphische Auswertung der Daten von Durchlauf eins und zwei, erstellt mit einer kleinen Java-Applikation, die einige Informationen der aufgezeichneten XML-Datei graphisch darstellt. Dabei repräsentiert die vertikale Achse die momentane Abweichung zur Ideallinie, die nach oben hin zu nimmt. Die horizontale Achse stellt von links nach rechts den zeitlichen Verlauf dar. Die vertikalen schwarzen Striche markieren den Abschluss einer Runde. Ist die Linie grün, hat der Roboter zu dem Zeitpunkt mit hoher Geschwindigkeit seine Position verändert, ist sie rot, hat der Roboter gestoppt, oder sich langsam bewegt. Der obere Graph in Abbildung 19 zeigt den Verlauf des ersten Durchlaufs im Uhrzeigersinn. Darunter wird der Durchlauf auf der gleichen Strecke gegen den Uhrzeigersinn dargestellt.

Hier wird zunächst deutlich, dass der Roboter im Uhrzeigersinn nicht näher als ca. einen Zentimeter an die Ideallinie gekommen ist. Der Grund hierfür ist der, dass er den äußeren Rand des Kurses abgefahren hat, die Ideallinie jedoch am inneren Rand liegt. Die Linie hat eine Breite von ca. einen Zentimeter. Auch hier lässt es sich gut erkennen, dass der Linefollower im ersten Durchlauf in der dritten Runde bei der zweiten Kurve kurzzeitig die Linie verloren hatte. Die Stelle ist mit einem Kreis markiert.

Betrachtet man den Graphen des zweiten Durchlaufs des Experiments gegen den Uhrzeigersinn auf dem einfachen Kurs, sieht man auch deutlich, dass der Roboter in der ersten und zweiten Kurve, die enger gestaltet sind, in jeder Runde Probleme bekam dem Kurs zu folgen (markierte Stellen), wohingegen ihm die Kurven mit weiteren Radius in der Geschwindigkeit kaum verlangsamten.

Der dritte und vierte Versuchsdurchlauf wurde auf einer modifizierten Version des einfachen

Kurses (siehe Kapitel 4.1.1) durchgeführt. Dabei sollte der Linefollower diesen komplexeren Kurs einmal im und einmal gegen den Uhrzeigersinn über 5 Runden abfahren.

Der Linefollower hatte im Vergleich zum einfach Kurs, im Uhrzeigersinn abgefahren, deutlich größerer Schwierigkeiten den Kurs zu bewältigen. Obwohl sich die Distanz einer Runde zum einfach Kurs durch die beiden zusätzlichen Kurvenelemente nur leicht erhöht hat (siehe Abbildung 20), benötigt der Roboter pro Runde durchschnittlich 35.4812 Sekunden, also mehr als doppelt so lange, wie im einfachen Kurs. Auch die mittlere und maximale Abweichung ist um ca. 0.7 Zentimeter höher als im ersten Durchlauf.

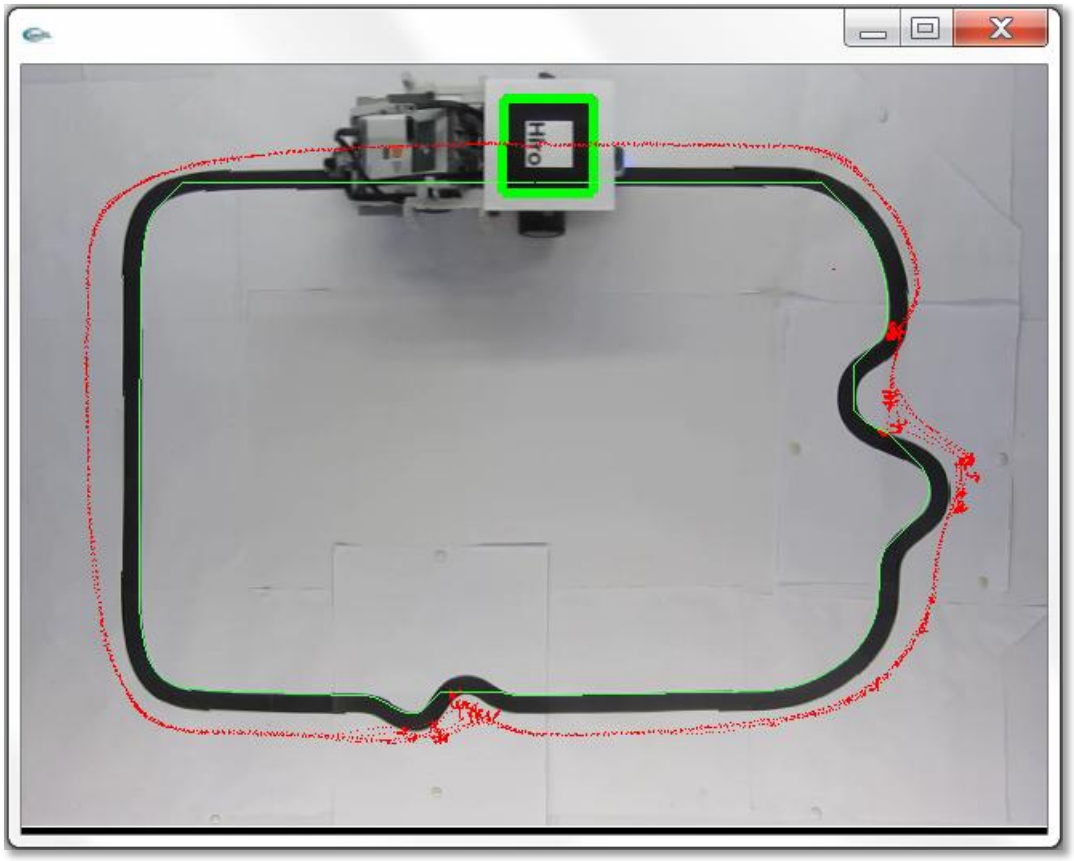

Abbildung 20 - aufgezeichneter Kurs des Linefollowers nach fünf Runden im komplexeren Kurs im Uhrzeigersinn

Komplexer Kurs	mittlere Abweichung	max. Abweichung	mittlere Rundenzeit
Uhrzeigersinn	2.7226 cm	5.0152 cm	35.4812 sec

Komplexer Kurs	Runde 1	Runde 2	Runde 3	Runde 4	Runde 5	Gesamtzeit
Uhrzeigersinn (in sec)	38.672	34.415	33.45	32.768	38.101	**177.406**

Abbildung 21: tabellarische Auswertung der Metriken im komplexen Kurs im Uhrzeigersinn gefahren

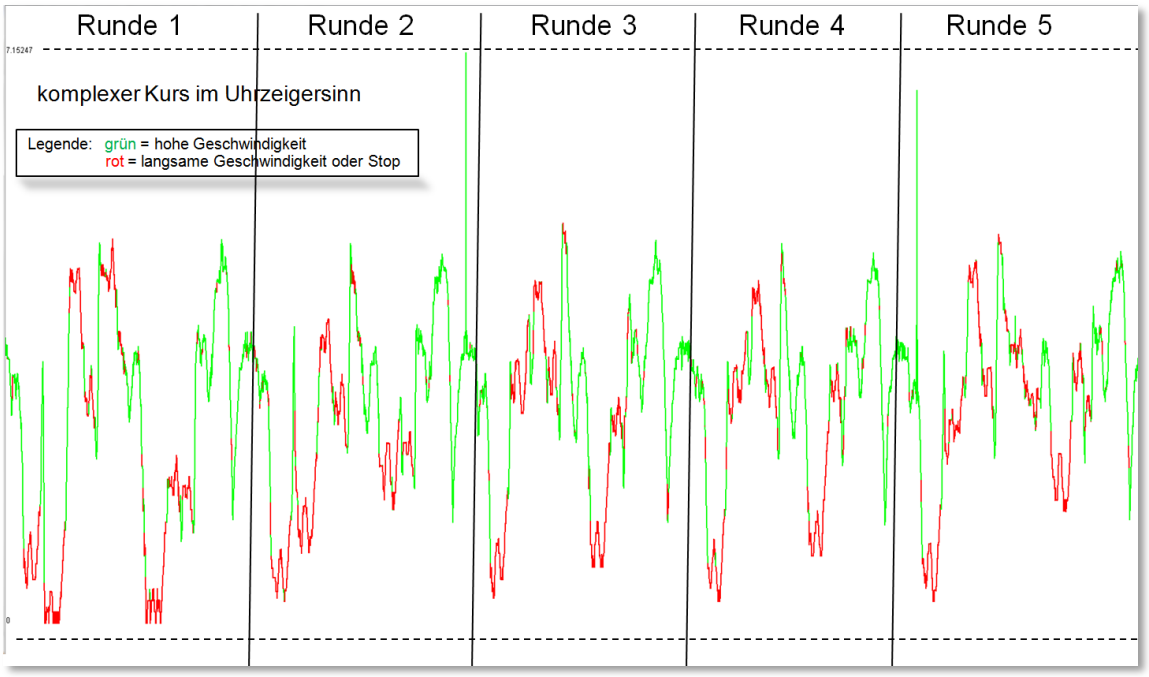

Abbildung 22: graphische Auswertung des komplexen Kurses, im Uhrzeigersinn gefahren

Bei der Betrachtung der graphischen Auswertung (siehe Abbildung 22), erkennt man deutlich, dass der Roboter wesentlich häufiger stoppen musste, um die Linienspur wiederzufinden. Dies veranschaulichen die vielen roten Bereiche im Graphen, die sich auch in Abbildung 20 an den Punktehaufen wiederspiegeln. So bereitet dem Roboter die erste Schikane die größten Probleme, aber auch die darauf folgende Schikane hielt den Linefollower merklich auf. Die beiden letzten engeren Kurven, bewältigte er identisch wie im ersten Versuchsdurchlauf, was im Vergleich der Graphen zwischen der Abbildung 16 und der Abbildung 20 an deren sehr ähnlichen Form und Färbung kurz vor Vollendung einer Runde zu erkennen ist.

//TODO: evtl. Graphen besser diskutieren und Zusammenhang mit Draufsicht herstellen?

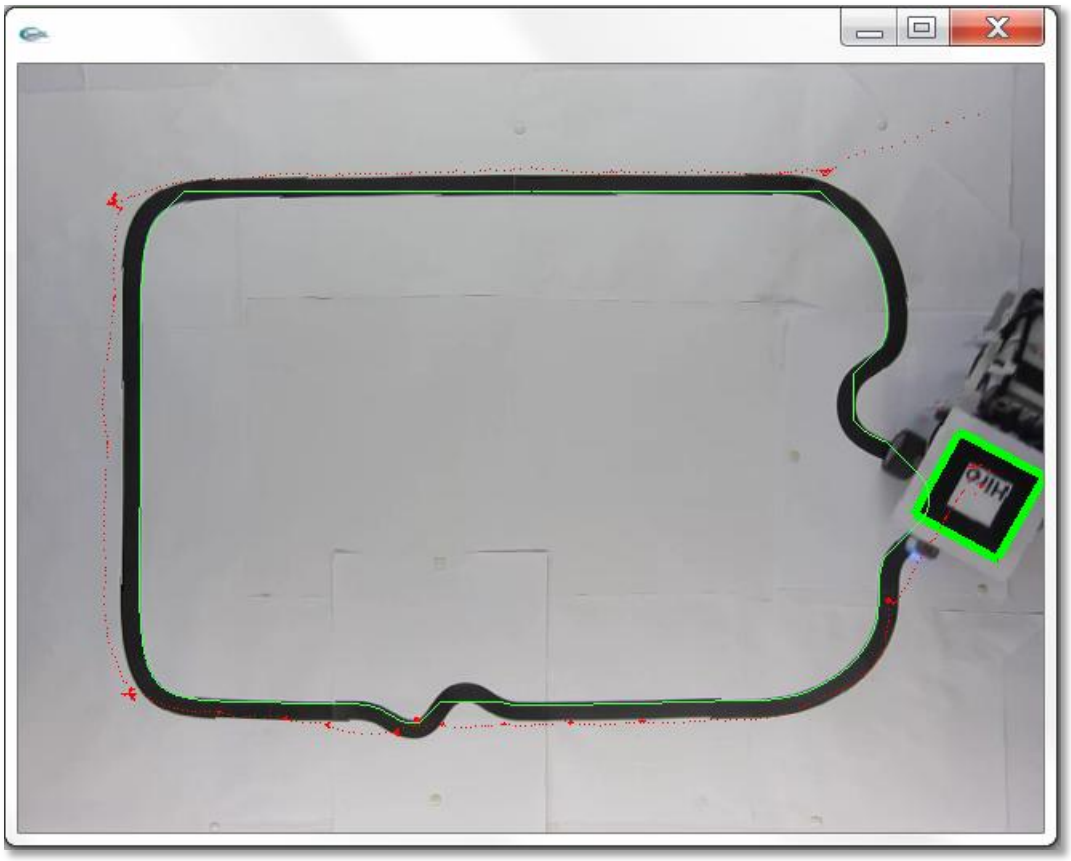

Abbildung 23 - aufgezeichneter Kurs des Linefollowers nach knapp einer Runde gegen den Uhrzeigersinn; nun wird er den Kurs im Uhrzeigersinn aufnehmen

Beim vierten Durchlauf des Experiments sollte der Linefollower den komplexeren Kurs, für fünf Runden gegen den Uhrzeigersinn abfahren. Dieser Durchlauf wurde fünfzehn mal wiederholt, doch leider konnte der Roboter bei keinem dieser Versuche auch nur eine Runde beenden, sondern zeigt stets das gleiche Verhalten. In der ersten Kurve der zweiten Schikane verlor er jedes Mal die Linie und begann daraufhin rechtsseitig zu suchen. Die Implementierung des Roboters ist bei Verlust der Linie so gestaltet ist, dass er bei der Suche rechtsseitig orientiert. So führen die aufschaukelnden Bewegungen insgesamt zu einer Rechtsdrehung des Roboters. Damit trifft er bei dieser Schikane nach einer 180° Drehung wieder auf den Kurs und folgt ihm in gegen gesetzter Richtung, also im Uhrzeigersinn (siehe Abbildung 23). Dies lässt also keine Vergleichbaren Metriken zum Durchlauf drei zu.

4.4 Fazit

Eine aufwendigere Gestaltung des Versuches würde freilich zu noch präzisere Daten führen. Eine höher platzierte Kamera reduziert noch weiter die objektivbedingte perspektivische Verzeichnung. Eine höhere Videoauflösung liefert eine exaktere Repräsentation der Umgebung und damit auch der Position des Roboters. Eine höhere Bildaufzeichnungsfrequenz liefert eine größere Anzahl von Messpunkten. Um diese Daten auswerten zu können ist allerdings auch die dementsprechend leistungsstarke Hardware Voraussetzung, da das Videomaterial in Echtzeit vom Programm ausgewertet wird, selbst wenn es sich nur um eine Videoaufzeichnung handelt. Bei dieser Arbeit mussten leider aus den finanziellen und örtlichen Begebenheiten Kompromisse eingegangen werden.

Dennoch zeigt das durchgeführte Experiment exemplarisch, dass es lediglich mit einer Kamera und dem Programm Robotracker möglich ist, sehr exakte Daten über eine Experimentalumgebung und der Position eines Markers zu gewinnen und diese über beliebige Metriken zu verarbeiten.

Die Ergebnisse der beiden Durchläufe im einfachen Kurs deuteten schon darauf hin, dass die Implementierung des Linefollowers auf Rechtskurven, bzw. Kurse im Uhrzeigersinn optimiert ist. Dies bestätigten die gesammelten Daten und Erfahrungen bei den Durchläufen im komplexeren Kurs.

Die durch den *RoboTracker* gewonnenen und ausgewerteten Daten lassen den Schluss zu, dass der verwendete Linefollower rechtskurvenlastige Kurse schneller folgen kann. Die Kurven sollten nicht zu scharf gestaltet sein, insbesondere Linkskurven, da der Linefollower ansonsten seine Orientierung auf dem Kurs verlieren kann.

5 Konklusion und Ausblick

Mit dem in dieser Arbeit vorgestellten Framework *RoboTracker*, wurde ein Werkzeug zur videogestützten Analyse von Experimentalumgebungen sowie der Positionsbestimmung eines in ihr befindlichen Markers entwickelt. Es lässt sich aufgrund der Verwendung des Fassadenpatterns mit geringem Aufwand an die unterschiedlichen Ansprüche von Experimenten anpassen. Zusätzlich bietet es die Möglichkeit anhand beliebig definierbarer Metriken die aufgezeichneten Daten auszuwerten. Bereits die vorhandene Implementierung ist aufwandslos dazu in der Lage ein von oben betrachtetes Labyrinth zu erkennen, den kürzesten Pfad vom Start zum Ziel zu berechnen und den Weg eines darin befindlichen, mit einem Marker ausgestatteten Roboters aufzuzeichnen.

Um die Möglichkeiten und die Leistungsfähigkeit des Frameworks noch zu erweitern, wäre es zum Beispiel denkbar das in dieser Arbeit verwendete *ARToolkit*-Framework durch eine modernere, effizientere Weiterentwicklung auszutauschen. Ein erweitertes graphisches Benutzerinterface würde zu einer effizienteren Nutzbarkeit des Frameworks beitragen. Denkbar ist auch der Einsatz einer Kamera mit einer größeren Auflösung oder einer höheren Bildaufzeichnungsrate. Dies setzt allerdings in der momentan vorhandenen Implementierung eine leistungsstarke Hardware vor raus, wenn man die Videos in Echtzeit analysieren will. Aufgezeichnete Videos könnten entsprechend verlangsamt werden, damit die Hardware noch in der Lage ist die Daten zu erfassen. Eine Offlineanalyse, die aufgezeichnete Videos nicht in Echtzeit auswertet wäre ein weiterer Ansatz, dieser Problematik entgegenzutreten. Auch die Unterstützung mehrerer Kameras, die simultan zusammenarbeiten, würde Interessante neue Möglichkeiten bieten und die Stabilität der Markererkennung und deren exakten Positionsbestimmung weiter verbessern. Nützlich wäre zudem die Möglichkeit den die tatsächliche Position von Markern softwareseitig zu manipulieren, und damit den Meßpunkt zu verschieben. So könnte zum Beispiel die Position eines Roboters unter seinem Boden aufgezeichnet werden.

Die Unterstützung einer simultanen Erkennung mehrerer Marker wird dann interessant, sobald bei einem Experiment mehr als ein bewegliches Objekt und dessen Position von Interesse ist. Das hier verwendete *ARToolkit* bietet dafür schon die notwendigen Voraussetzungen.

Denkbar wäre auch eine vollkommen andere Nutzung, der in Echtzeit gesammelten Daten. So könnte man einem Roboter seine durch das Framework ermittelte Position, Informationen über seine Umgebung und die daraus berechneten Daten (z.B. die Ideallinie durch ein Labyrinth) übermitteln, um ihm die Navigation bzw. das Erreichen seines Ziels zu erleichtern.

6 Literaturverzeichnis

[1] RoboCup URL: http://www.robocup.org (27.02.2012)

[2] The Scribbler URL: http://www.parallax.com/ScribblerFamily/tabid/825/Default.aspx (27.02.2012)

[3] Frank Zimmermann, Modellgetriebene Softwareentwicklung für Lego NXT, No 2008-01, Arbeitspapiere der Nordakademie from Nordakademie - Hochschule der Wirtschaft, 2008, Seite 2

[4] Open Source: leJOS Project Page. Februar 2012, URL: http://lejos.sourceforge.net. (25.02.2012)

[5] Open Source: nxtOSEK, URL: http://lejos-osek.sourceforge.net/ (25.02.2012)

[6] Lego Mindstorms–Linien folgen URL: http://www.ptec-media.de/files/line-follower-doku.pdf (25.02.2012)

[7] Augmented Reality Toolkit URL: http://www.hitl.washington.edu/artoolkit/ (24.01.2012)

[8] ARToolkit Plus URL: http://handheldar.icg.tugraz.at/artoolkitplus.php (24.02.2012)

[9] ARToolKitPro - ARToolworks URL: http://www.artoolworks.com/products/stand-alone/artoolkitpro/ (26.02.2012)

[10] osgART URL: http://www.osgart.org/wiki/index.php/Main_Page (26.02.2012)

[11] Studierstube URL: Tracker http://handheldar.icg.tugraz.at/stbtracker.php (26.02.2012)

[12] Mark Fiala, ARToolkit Applied to Panoramic Vision for Robot Navigation URL: http://www.cipprs.org/papers/VI/VI2003/papers/S3/S3_fiala_181.pdf (27.02.2012)

[13] Canon Powershot SX230 HS – Specifications URL: http://www.canon.de/For_Home/Product_Finder/Cameras/Digital_Camera/PowerShot/PowerShot_SX230_HS/index.aspx?specs=1

{Effiziensvergleich toolkits: URL: http://www.sciweavers.org/read/visual-marker-detection-and-decoding-in-ar-systems-a-comparative-study-151445 }

7 Bedienungsanleitung

// Installationsanleitung

// Bedienung von roboTracker

Appendix

In den Appendix kommen längere Code-Ausschnitte und anderes Material, das nicht in den Textfluß passt. Man kann diesen in Abschnitte unterteilen (zB Appendix A, Appendix B, …) um darauf im Text zu verweisen.