ALBERT-LUDWIGS-UNIVERSITÄT
FREIBURG
INSTITUT FÜR INFORMATIK

Machine Learning Lab
Prof. Dr. Martin Riedmiller

Intelligente Objekterkennung für ein lernfähiges Carrerabahn-System

Bachelorarbeit

Jonas Gehring

Erstgutachter:	Prof. Dr. Martin Riedmiller
Zweitgutachter:	Dr. Cyrill Stachniss
Betreuer:	Stefan Welker
Abgabedatum:	21. Oktober 2009

ERKLÄRUNG

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Ort, Datum

Unterschrift

Kurzfassung

Um eine robuste Objekterkennung in bildverarbeitenden, alltagstauglichen Anwendungen zu gewährleisten, ist meist hochgradiges Fachwissen des jeweiligen Entwicklers gefordert. In der vorliegenden Arbeit wird nun ein robustes, effizientes und nahezu autonomes System zur optischen Positionsbestimmung von Objekten entworfen. Anstelle von Fachwissen wird auf Techniken des maschinellen Lernens zurückgegriffen, um im Rahmen des von Riedmiller et al. entwickelten *NeuroRacer* verschiedene Rennwagen auf einer Carrerabahn zu erkennen.

Hierbei werden die Bilder, aus denen einzelne Objekte zunächst segmentiert werden, von einer sich über der Strecke befindlichen Kamera aufgezeichnet. Für die Bereitstellung des zur Segmentierung verwendeten Hintergrundes wird ein neuer Ansatz vorgestellt, der die Ergebnisse der Klassifikation berücksichtigt. Ferner wird beschrieben, wie eine automatisierte Erfassung der Trainingsdaten gestaltet werden kann, die für die als Klassifikatoren verwendeten neuronalen Netzwerke benötigten werden.

Inhaltsverzeichnis

1. Einführung **9**
1.1. Einführung . . . 9
1.2. Die Carrera Autorennbahn . . . 9
1.3. Das NeuroRacer-System . . . 10
1.4. Carrera Vision . . . 10
1.5. Anforderungsanalyse . . . 12
1.6. Entwurf . . . 12
1.6.1. Segmentierung . . . 12
1.6.2. Klassifikation . . . 12
1.6.3. Generierung des Hintergrundbildes . . . 13

2. Segmentierung **13**
2.1. Generierung des Differenzbildes . . . 13
2.2. Extraktion von Objekten . . . 15
2.3. Ausgerichtete umschließende Rechtecke . . . 15

3. Akquirierung von Trainingsdaten **17**
3.1. Klassifikation anhand eines Referenzobjekts . . . 17
3.1.1. Merkmale . . . 17
3.2. Auswahl des Referenzobjekts . . . 19

4. Training der Klassifikatoren **19**
4.1. Netzwerkeingabe . . . 19
4.2. Netzwerktopologie . . . 20
4.3. Training . . . 21

5. Klassifikation **21**
5.1. Objektklassifikation . . . 21
5.2. Iterative Hintergrundgenerierung . . . 22

6. Ergebnisse **23**
6.1. Differenzbasierte Segmentierung . . . 23
6.2. Automatisierte Erfassung von Trainingsdaten . . . 23
6.3. Iterative Hintergrundgenerierung . . . 24
6.4. Evaluation der Klassifikatoren . . . 24
6.5. Laufzeit der Klassifikation . . . 25
6.6. Vergleich mit Carrera Vision . . . 26

7. Experimente **26**
7.1. Einfluss der Trainingsdaten . . . 26
7.2. Einfluss der Netzwerkeingabe . . . 27

8. Schlussfolgerungen **28**
8.1. Einsatz in Carrera Vision 28
8.2. Weitere Anwendungsszenarien 29
8.3. Mögliche Verbesserungen 29

A. Berechnung der ausgerichteten umschließenden Rechtecke **31**

B. Künstliche Bildverfälschung **32**
B.1. Aufhellung 32
B.2. Abdunklung 32
B.3. UV-Verschiebung 32
B.4. Einfügen eines Fremdobjekts 33

C. Implementation **33**
C.1. Verwendete Technologien 33
C.2. Wissensdatenbank 34
C.3. Graphische Benutzerschnittstelle 34
C.3.1. Hauptfenster 34
C.3.2. Akquirierung von Trainingsdaten 34
C.3.3. Training der Netzwerke 34
C.3.4. Klassifikation 37

1. Einführung

1.1. Einführung

Geräte zur digitalen Aufzeichnung von Bilddaten sind inzwischen zu Alltagsgegenständen geworden; ein Trend, der sich in Zukunft höchstwahrscheinlich fortsetzen wird. Zugleich werden bildverarbeitende Systeme in einer Vielzahl verschiedener Bereiche eingesetzt, sei es nun in Form eines Barcodescanners im Supermarkt oder etwa zur automatischen Gesichtserkennung in Digitalkameras. Die Implementierung von Systemen für diese Einsatzgebiete setzt oft ein hochgradiges Fachwissen von Seiten der Entwickler voraus, da die Benutzung im Alltag sowohl robuste als auch effiziente Algorithmen verlangt. Zugleich sollte ein solches System im optimalen Fall keinerlei manuelle Nachjustierung im laufenden Betrieb erfordern. Es sollte daher für den Benutzer einer Kamera nicht nötig sein, die Parameter der Gesichtserkennung von Hand anzupassen, falls sich beispielsweise die Eigenschaften der Umgebung ändern oder eine weitere Person ins Blickfeld tritt. Eine in der Praxis zwangsweise auftretende Situation, die bei farbbasierten Algorithmen häufig Ursache für schlechte Resultate ist, ist eine Änderung der Lichtverhältnisse.

Die Hauptmotivation dieser Arbeit ist es daher, ein System zu entwickeln, dass einerseits ohne großes Fachwissen bezüglich des Einsatzgebiets implementiert werden kann und andererseits weitgehend autonom arbeitet. Die erste dieser Eigenschaften wird mittels Algorithmen aus der informationstechnischen Disziplin des maschinellen Lernens erreicht. Dies versetzt das System weiterhin in die Lage, auch in bisher unbekannten Situationen die nötige Funktionalität zu gewährleisten.

Um die Eigenschaft der autonomen Ausführung zu gewährleisten, soll trotzdem so viel Vorwissen wie möglich in die Entwicklung der Algorithmen einfließen – allerdings nicht in Form von genauen Justierungen von Parametern, sondern vielmehr durch Beschränkung auf ein spezielles Einsatzgebiet. Hierdurch können elementare Zusammenhänge direkt aus der jeweiligen Domäne abgeleitet werden und für den Entwurf der Einzelkomponenten genutzt werden. Diese wird nun zunächst ausführlich beschrieben.

1.2. Die Carrera Autorennbahn

Autorennbahnen von Carrera erfreuen sich bei Kindern und zahlreichen Enthusiasten großer Beliebtheit. Eine Rennstrecke besteht aus einzelnen Modulen (Abb. 1), die beliebig kombinierbar sind. Die Rennwagen sind spurgeführt und werden von einem Elektromotor angetrieben, der die nötige elektrische Energie über die jeweilige Spurrille erhält. Durch die festen Spurrillen kann der Benutzer der Rennbahn lediglich die Geschwindigkeit seines Rennwagens kontrollieren. Hierzu dient ein Controller, der über einen Widerstand den Stromfluss auf der Strecke regelt.

Die Rennwagen können ihre Spur während der Fart nur verlassen, indem sie durch zu hohe Zentrifugalkräfte die Verankerung lösen. Diese meist in der vorderen Hälfte des Wagens angebrachte Verankerung ist drehbar gelagert, so dass das Heck des Wagens permanent ausbricht.

[1] `http://www.carrera-toys.de`

Abbildung 1: Einzelteile einer Autorennbahn von Carrera[1].

1.3. Das NeuroRacer-System

Der von Riedmiller et al. entwickelte *NeuroRacer* ist ein System, dass einen Rennwagen auf einer Carrerabahn steuern kann. Es verwendet das ebenfalls von Riedmiller et al. entworfene *CLSquare*-Framework, um durch optimierendes Lernen eine Strategie zur Steuerung zu entwickeln [1]. Eine Kamera, die über der Strecke angebracht ist, liefert die für die Positionsbestimmung des Fahrzeugs verwendeten Bilddaten (Abb. 2).

Intern wird die Strecke durch ein eindimensionales Modell repräsentiert, bei dem die aktuelle Position des Wagens im Intervall $[0, 1]$ dargestellt wird.

1.4. Carrera Vision

Zur Erfassung der Position des Rennwagens auf der Strecke wird das von Sascha Lange entwickelte *Carrera Vision* verwendet. Zur Segmentierung des Fahrzeugs wird ein farbbasiertes Verfahren genutzt [2]. Grundlage hierfür bildet eine Funktion

$$s : (r, g, b) \rightarrow c$$

die jeden Punkt im RGB-Farbraum auf eine Klasse c abbildet. Vor Beginn der eigentlichen Positionsbestimmung muss s justiert werden, so dass der Rennwagen mittels einer einzigen Klasse klar segmentiert werden kann (Abb. 3). Die Funktion s ist, da sie auf die Farbe eines einzelnen Pixels beschränkt ist, von einer Vielzahl nicht direkt beeinflussbarer Parameter abhängig. Als Beispiel seien hier die unterschiedlich gefärbten Rennwagen oder die momentane Beleuchtung angeführt.

Neben der angesprochener Erkennung enthält *Carrera Vision* ebenfalls Funktionen, um die aktuelle Position des Wagens wie in Abschnitt 1.3 angegeben zu parametrisieren. Hierzu wird vor dem eigentlichen Betrieb zunächst der Streckenverlauf aufgezeichnet, indem der Rennwagen die Strecke vollständig abfährt.

Neben der aktuellen Position werden im laufenden Betrieb weiterhin die Abweichung des Rennwagens von der Strecke und dessen Winkel gegenüber dem normalen Streckenverlauf berechnet. Somit kann auch negatives Feedback, d.h. der Rennwagen befindet sich nicht mehr auf der Strecke, an *NeuroRacer* gesendet werden.

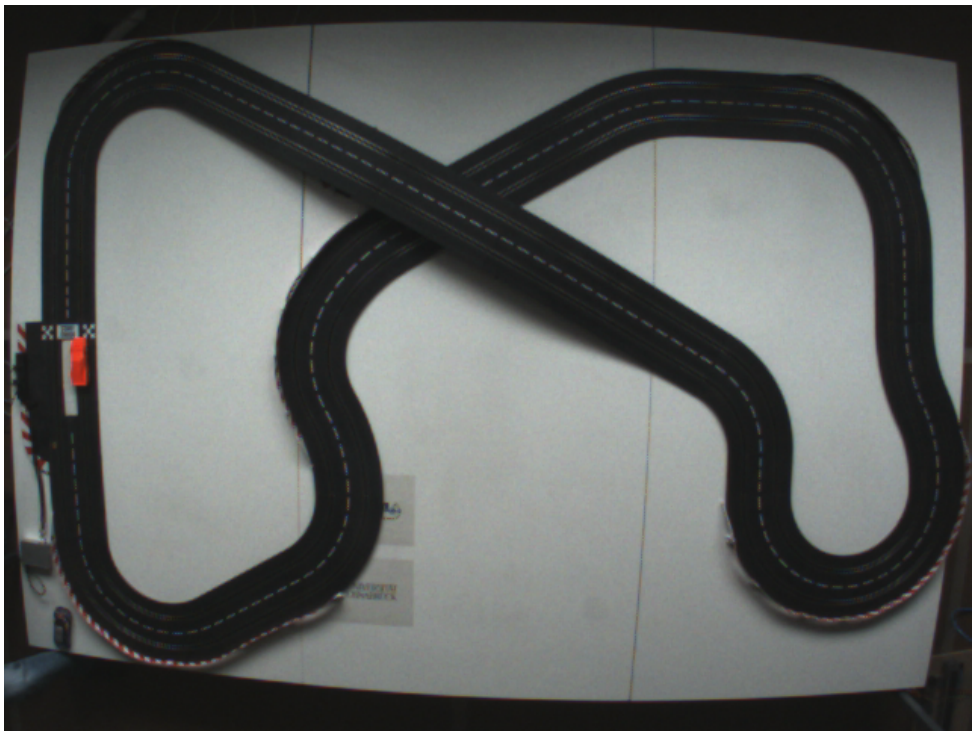

Abbildung 2: Das von der Kamera aufgenommene Bild der Rennstrecke. Links auf der Strecke befindet sich ein Rennwagen.

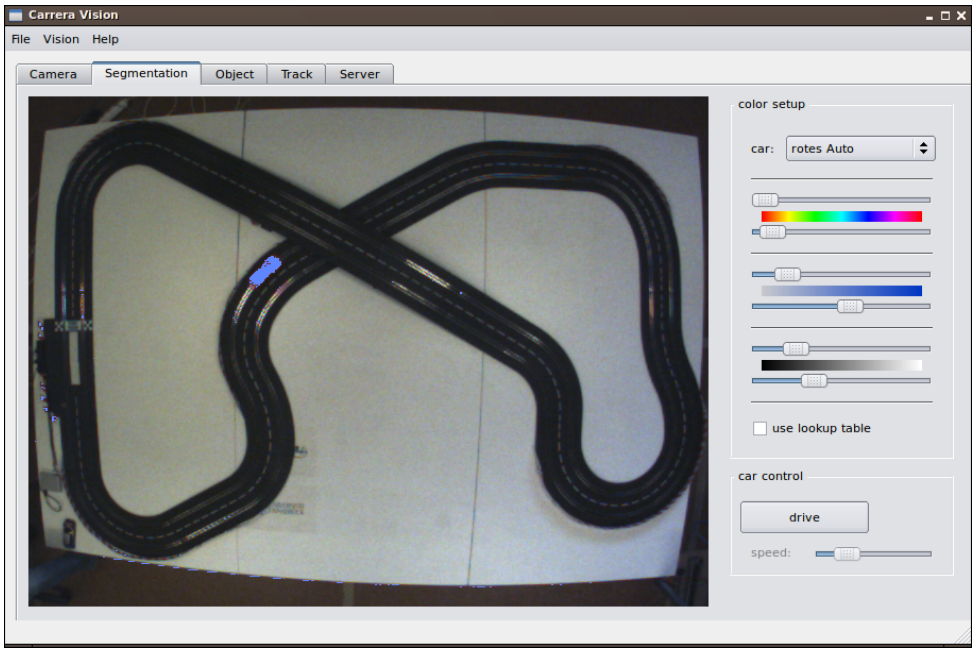

Abbildung 3: Das Benutzerinterface zur Farbsegmentierung von *Carrera Vision*.

1.5. Anforderungsanalyse

Die Anforderungen an das zu entwerfende System sind durch die oben beschriebene Funktionsweise der bereits vorhandenen Komponenten schon relativ klar umrissen. Es ist jedoch nicht Ziel dieser Arbeit, die komplette Funktionalität von *Carrera Vision* nachzubilden. Vielmehr soll eine alternative Lösung zur Segmentierung und Klassifikation der Rennwagen entworfen werden. Weiterhin soll die Möglichkeit des Einsatzes mehrerer Klassifikatoren für unterschiedliche Rennwagen bestehen. Dies ist für das *NeuroRacer*-System von Vorteil, da so beispielsweise die Rennwagen untereinander ausgetauscht werden könnten, ohne das ein komplettes Wiedererlernen der korrekten Steuerbefehle notwendig wäre.

Die Forderungen an die Laufzeit der Erkennung sind relativ hoch angesetzt. Die Kamera über der Carrerabahn sendet Bilder der Strecke in einer Auflösung von 640 mal 480 Bildpunkten und im Abstand von 16.67 ms, arbeitet also mit einer Bildfrequenz von 60 Hz. Im Idealfall sollte die Rechenzeit für Segmentierung und Klassifikation also unter 16.67 ms betragen, um alle zu Verfügung stehenden Informationen an *NeuroRacer* weiterzureichen und eine akzeptable Reaktionsgeschwindigkeit der Steuerung zu erreichen.

1.6. Entwurf

Basierend auf den oben definierten Anforderungen wird nun ein System entworfen, dass sowohl die Segmentierung als auch die Klassifikation der Rennwagen beinhaltet.

1.6.1. Segmentierung

Um Objekte auf dem von der Kamera aufgezeichneten Bild zu segmentieren wird ein differenzbasierter Ansatz gewählt, der in der Literatur unter der Bezeichnung *Background Subtraction* bekannt ist. Hierfür wird vom aktuellen Bild ein zuvor gespeichertes Hintergrundbild subtrahiert. Das Ergebnis wird mit Hilfe eines Mindestbetrags pro Bildpunkt in ein Binärbild transformiert.

Das Binärbild fungiert nun also als Maske, welche Vordergrund und Hintergrund differenziert. Aus den Bildpunkten, die den Vordergrund darstellen, werden die einzelnen Objekte segmentiert.

Eine ausführliche Beschreibung der zur Segmentierung verwendeten Algorithmen befindet sich in Kapitel 2.

1.6.2. Klassifikation

Als Klassifikatoren für die einzelnen Rennwagen werden neuronale Netzwerke gewählt. Sie stellen mächtige Instrumente des maschinellen Lernens dar und wurden bereits in einer Vielzahl von Anwendungsfällen, unter anderen auch zur Objektklassifikation, eingesetzt [3].

Wie bei einer Vielzahl von Algorithmen aus der Disziplin des Maschinellen Lernens wird hier vorausgesetzt, dass das System Zugang zu Beispieldaten besitzt. Diese werden benötigt, um die Netzwerke im Vorfeld der Klassifikation zu trainieren. In dieser Arbeit

wird daher eine Methode zur größtenteils automatisierten Beschaffung von Trainingsdaten vorgestellt, so dass dies auch im realen Betrieb bequem vorgenommen werden kann.

1.6.3. Generierung des Hintergrundbildes

Während eines Rennens kann es freilich vorkommen, dass sich die Lichtverhältnisse auf der Strecke ändern oder neue Objekte im Sichtfeld der Kamera auftauchen, die keine Rennwagen darstellen. Dies hätte unter Umständen dramatische Auswirkungen auf die Segmentierung der eigentlich zu klassifizierenden Objekte. Eine ständige Anpassung des verwendeten Hintergrunds ist im realen Betrieb daher unumgänglich. Für diese Zwecke wird eine neue Methode zur iterativen Generierung eines Hintergrundbildes vorgestellt, die auf die Ergebnisse der Klassifikation des letzten Kamerabildes zurückgreift.

2. Segmentierung

2.1. Generierung des Differenzbildes

Zur Segmentierung von Objekten in einem von der Kamera übertragenen Bild wird wie in Abschnitt 1.6.1 beschrieben zunächst ein Differenzbild berechnet. Neben dem aktuellen Bild K wird zur Differenzbildung zusätzlich ein Hintergrundbild B benötigt.

Für die Segmentierung zur Aufnahme von Trainingsdaten wird ein statisches Hintergrundbild verwendet. Dieses wird jeweils im Vorfeld aus mehreren aufeinanderfolgenden Bildern generiert. Dies erfolgt online, so dass das Hintergrundbild nach der Initialisierung mit einem beliebigen Bild laufend aktualisiert wird. Hierzu wird ein von der Kamera empfangenes Bild K mit einem Parameter α gewichtet und auf das vorige Hintergrundbild B' addiert. Es gilt also für jeden Bildpunkt $b \in B, k \in K$:

$$b = (1 - \alpha) \cdot b' + \alpha \cdot k \tag{2.1}$$

Um ein aussagekräftiges Hintergrundbild zu erhalten, dass die Information vieler Einzelbilder speichert, sollte für α ein positiver Wert nahe 0 gewählt werden (hier: $\alpha = 0.01$).

Die eigentliche Differenzbildung erfolgt nun, indem ein aktuelles Bild K vom Hintergrundbild B subtrahiert wird, wobei zusätzlich ein Grenzwert t verwendet wird. Somit wird die Robustheit im Bezug auf leichte Schwankungen der Farbwerte, welche in der Praxis zwangsläufig auftreten, erhöht.

$$f_{diff}(b, k) = \begin{cases} 1, & \text{falls } b - k \geq t \\ 0, & \text{sonst} \end{cases} \tag{2.2}$$

Das Resultat der Differenzbildung D ist nun ein binäres Bild. Jeder darin enthaltene Bildpunkt d ergibt sich aus Anwendung von f_{diff} auf die der Position im Bild entsprechenden Bildpunkte des aktuellen Bildes K und des Hintergrundbildes B.

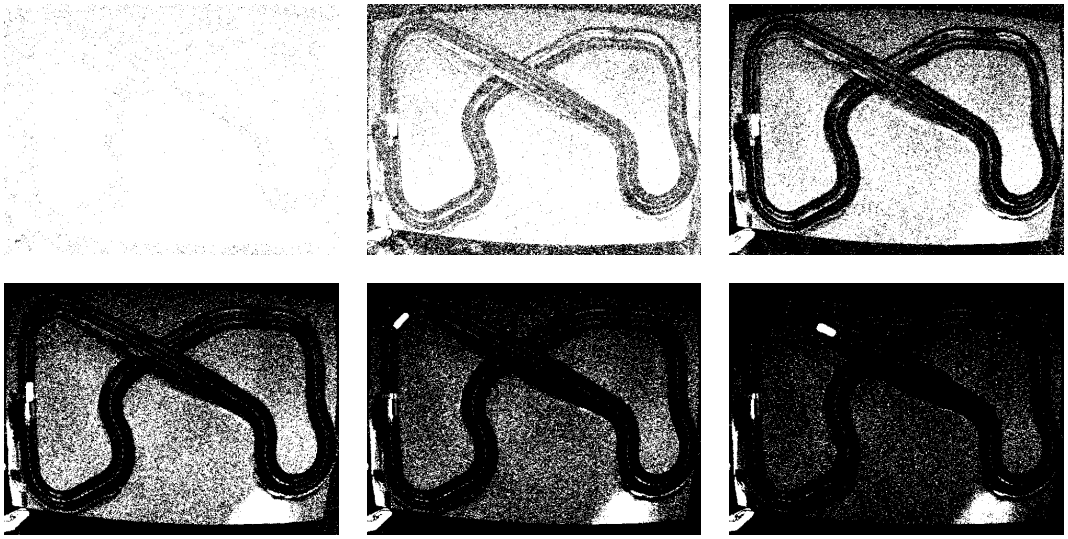

Abbildung 4: Anpassung des Grenzwertes für die Berechnung des Differenzbildes. Zwischen den hier abgebildeten Einzelbildern liegt jeweils noch ein weiteres Bild.

Der Grenzwert t wird nach jeder Differenzbildung automatisch angepasst. Als Kriterium für die Justierung wird der prozentuale Anteil der in D gesetzten Bildpunkte verwendet:

$$P_{set} = \frac{|\{d \in D : d = 1\}|}{|D|} \tag{2.3}$$

Ausgehend von einer unteren und einer oberen Schranke wird der Grenzwert t erhöht oder verringert, wenn P_{set} unterhalb der unteren bzw. oberhalb der oberen Schranke liegt. Abbildung 4 zeigt den initialen Vorgang der Justierung, der mit $t = 1$ gestartet wird.

Als robuste Grenzwerte für P_{set} haben sich in den zahlreichen Testläufen 0.01 für die untere und 0.03 für die obere Schranke herausgestellt.

Generierung eines dynamischen Hintergrundbildes Während die Verwendung eines statischen Hintergrundbildes für die Erfassung von Trainingsdaten ausreichend ist, muss das Hintergrundbild im laufenden Betrieb ständig angepasst werden (siehe Abschnitt 1.6.3). Ohne diese Anpassung ist keine robuste Klassifikation möglich, da beispielsweise bereits eine geringfügige Änderung der Lichtverhältnisse die Differenzbildung negativ beeinflussen kann.

Die hier entwickelte Technik zur iterativen Berechnung des Hintergrundbildes wird in Abschnitt 5.2 vorgestellt.

2.2. Extraktion von Objekten

Aus dem Differenzbild werden nun die verschiedenen Objekte mittels einer Variante des *Region Growing* extrahiert [4]. Bei dieser Segmentierungstechnik werden zwei Bildpunkte als der selben Region zugehörig betrachtet, falls sie eine Nachbarschaftsrelation erfüllen. Zusätzlich werden oft noch weitere Anforderungen gestellt, die die Ähnlichkeit beider Bildpunkte sicherstellen sollen. Für ein Graustufenbild kann hierfür beispielsweise eine Höchstdifferenz der Grauwerte verwendet werden. Regionen werden durch iteratives Vergrößern gebildet, in dem von einem oder mehreren Startpunkten ausgegangen wird. Anschließend wird für die an die Region angrenzende Bildpunkte anhand der beschrieben Relationen entschieden, ob sie ebenfalls Teil der Region sind.

In dieser Arbeit wird ein Vorkommen der Bildpunkte im Differenzbild als einziges weiteres Ähnlichkeitsmaß verwendet. Dies ist zur Abgrenzung von einzelnen Objekten im Vordergrund ausreichend: befinden sich zwei Objekte in ausreichendem Abstand zu einander, sind sie durch den dazwischenliegenden Hintergrund im Differenzbild bereits getrennt.

Um die einzelnen Regionen zu extrahieren, wird das Differenzbild nun sequentiell abgesucht. Der *Region Growing*-Algorithmus wird für jeden im Differenzbild gesetzten Bildpunkt gestartet, falls dieser bislang noch keiner Region zugeordnet wurde.

Es ist weiterhin zu beachten, dass aus dem Differenzbild meist viele extrem kleine Regionen extrahiert werden, die nur wenige Bildpunkte enthalten. Diese Regionen werden in der Regel durch Rauschen erzeugt und können somit vernachlässigt werden. Da weiterhin die Laufzeit der Segmentierung durch Verarbeitung vieler kleiner Objekte unnötig zunehmen würde, wird eine Mindestgröße für segmentierte Regionen festgelegt.

Weiterhin ist bekannt, in welchen Größenordnungen sich die Rennwagen der Carrerabahn befinden. Somit kann ebenfalls eine Höchstgröße definiert werden. Es werden hier also schlussendlich nur Objekte segmentiert, deren an den Hauptachsen ausgerichtetes umschließendes Rechteck eine Breite und Höhe von mindestens 10 und höchstens 100 Bildpunkten besitzt.

2.3. Ausgerichtete umschließende Rechtecke

Um die extrahierten Objekte zu beschreiben, werden deren ausgerichtete umschließende Rechtecke (*Oriented Bounding Boxes*, kurz *OBB*) berechnet [5]. Gegenüber normalen, an den Hauptachsen des Koordinatensystems ausgerichteten Rechtecken sind diese besser zur Beschreibung von zweidimensionalen Objekten geeignet. Zum einen können durch die Ausrichtung oft kleinere und damit die Kontur besser beschreibende Rechtecke angegeben werden. Zusätzlich kann eine Aussage über die Lage des Objekts bezüglich der Hauptachsen gegeben werden. Dies wird unter anderem in der Klassifikation der Objekte durch neuronale Netzwerke genutzt, indem die Bilddaten als nahezu rotationsinvariante Eingabe betrachtet werden können (vgl. Abschnitt 4.1).

Für die hier vorgestellte Arbeit ist die Verwendung von Rechtecken zudem günstig, da sie die Form eines Rennwagens aus der Vogelperspektive gut beschreiben.

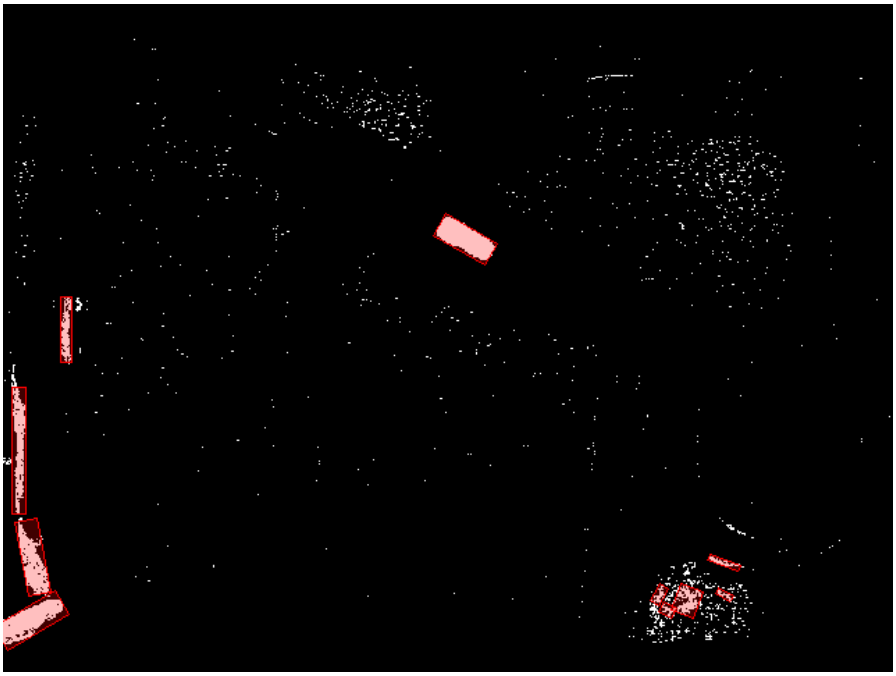

Abbildung 5: Mittels *Region Growing* extrahierte Objekte. Die jeweiligen ausgerichteten umschließende Rechtecke sind rot eingezeichnet. Ursache für die Objekte am linken unteren Rand ist u.a. das Kabel des Carrera-Controllers dar.

Zur Berechnung der minimalen OBB einer gegebenen Punktmenge sind in der Literatur bereits einige Algorithmen bekannt. Gottschalk et al. berechnen die Kovarianz-Matrix einer Punktmenge, deren Eigenvektoren die Ausrichtung des Rechtecks angeben [5]. Des Zentrums ergibt sich hier aus dem Durchschnitt aller Punktkoordinaten.

Lahanas et al. hingegegen beschreiben einen Algorithmus, der das minimale Rechteck durch Suche bestimmt [6]. Hier wird nach einer optimalen Rotation um alle Achsen des Koordinatensystems gesucht, die das Volumen des Rechtecks minimiert.

Für diese Arbeit ist einerseits keine allzu große Genauigkeit der OBB gefordert. Zum anderen handelt es sich um einen zweidimensionalen Fall, bei dem die Anzahl der Punkte zudem beschränkt ist. Daher wird ein schneller Brute-Force-Ansatz vorgestellt, der mögliche Ausrichtungen des Rechtecks mit einer zuvor festgelegten Schrittweite testet und die beste Lösung im Sinne eines Rechtecks mit minimalem Flächeninhalt zurückgibt. Der verwendete Algorithmus ist in Anhang A ausführlich beschrieben. Für die Implementation wurde eine Schrittweite von 10° gewählt. Dies mag auf den ersten Blick etwas grob erscheinen, wird den gestellten Anforderungen bezüglich Laufzeit und Genauigkeit jedoch gerecht.

3. Akquirierung von Trainingsdaten

Wie bereits erwähnt ist eines der Ziele des hier entworfenen Systems, eine weitgehend autonome Ausführung sicherzustellen und daher so wenig interaktive Benutzung wie möglich zu erfordern. Deshalb wird unter anderem großes Augenmerk auf die Erfassung von Trainingsdaten gelegt, so dass diese größtenteils automatisch von statten gehen kann.

3.1. Klassifikation anhand eines Referenzobjekts

Um Trainings- und Testdaten automatisiert zu erfassen, ist neben der Erkennung auch eine rudimentäre Klassifikation von Objekten nötig. Das Programm muss selbstständig und robust entscheiden, ob ein aufgenommenes Beispiel ein positives oder ein negatives Beispiel darstellt.

Wie sich herausstellte ist es bereits mit einfach zu berechnenden Merkmalen eines segmentierten Objektes möglich, eine gute Klassifikation zu erreichen. Es sei allerdings bereits im Voraus angemerkt, dass es sich hierbei um keine allzu robuste Klassifikation handelt. Dies ist angesichts des Einsatzgebietes jedoch keine signifikante Einschränkung. Die Trainingsdaten werden nicht im realen Betrieb aufgenommen, so dass Störungen wie beispielsweise zu viele sich verändernde Bereiche innerhalb des Bildes oder unterschiedliche Lichtverhältnisse minimiert werden können.

Die Klassifikation wird anhand einer Reihe von Merkmalen durchgeführt, die die Ähnlichkeit eines segmentierten Objekts zu einem zuvor definierten Referenzobjekt angeben. Hierzu wird pro Merkmal zunächst eine Rangliste aller Objekte aufgrund des jeweiligen Ähnlichkeitswertes erstellt. Für jedes Objekte werden anschließend dessen Ränge innerhalb der Listen gewichtet und summiert. Diese Summe stellt den endgültigen Differenzwert dar. Je kleiner der Betrag des Differenzwertes ist, desto ähnlicher ist ein Objekt dem Referenzobjekt. Das Objekt mit dem kleinsten Differenzwert wird als positives Beispiel, die restlichen Objekte werden als negative Beispiele markiert.

3.1.1. Merkmale

Unterschied des Histogramms Für die Berechnung des Histogramms wird für jeden Rot, Grün- und Blau-Farbwert dessen Häufigkeit in den sich im umschließenden Rechteck des Objekt befindenden Bildpunkten berechnet. Abschließend werden die Häufigkeiten logarithmisch skaliert, um die Aussagekraft des Histogramms zu verstärken. Ausgehend von einer konstanten Basis b kann folgende Funktion zur Skalierung des Histogramms verwendet werden:

$$f_b(x) = \log_b(1 + x \cdot (b - 1)) \tag{3.1}$$

Es gilt

$$x \in [0, 1] \Longrightarrow f_b(x) \in [0, 1] \tag{3.2}$$

sowie

$$f_b(0) = 0 \text{ und } f_b(1) = 1 \tag{3.3}$$

Für $b \to 1+$ gilt $f_b(x) = x$, wie sich mittels Differentiation zeigen lässt:

$$\begin{aligned}
\frac{d}{dx} f_b(x) &= \frac{d}{dx} \lim_{b \to 1+} \frac{\ln(1 + x \cdot (b-1))}{\ln b} \\
&= \frac{1}{\ln b} \cdot \frac{d}{dx} \ln(1 + x \cdot (b-1)) \\
&= \frac{1}{\ln b \cdot (1 + x \cdot (b-1))} \cdot \frac{d}{dx}(1 + x \cdot (b-1)) \\
&= \frac{b-1}{\ln b \cdot (1 + x \cdot (b-1))}
\end{aligned}$$

Es folgt mit (3.2) und (3.3)

$$\lim_{b \to 1+} \frac{d}{dx} f_b(x) = \lim_{b \to 1+} \frac{b-1}{\ln b} = 1 \Longrightarrow \lim_{b \to 1+} f_b(x) = x \tag{3.4}$$

Die logarithmische Verzerrung der Kurve nimmt mit steigendem b zu. Als Basis b wurde hier 100 gewählt, was einer relativ starken Skalierung entspricht.

Um das Histogramm weniger anfällig für leichte Schwankungen zu machen, werden die Häufigkeiten nicht für jeden der 256 möglichen Werte pro RGB-Farbkanal berechnet. Statt dessen wird die Bittiefe des Histogramms auf 5 Bit beschränkt. Allgemein kann eine Beschränkung des Histogramms auf $1 \leq k \leq 8$ Bit erreicht werden, indem statt des Wertes v eines Bildpunktes mit $0 \leq v \leq 255$ der Wert v' mit

$$v' = \left\lfloor \frac{v}{8 - k + 1} \right\rfloor$$

in der Berechnung der Häufigkeiten verwendet wird.

Um schließlich den Unterschied zwischen dem Histogramm des Objekts und dem Histogramm des Referenzobjekts zu berechnen, wird die absolute Differenz für jeden Eintrag berechnet. Diese wird über alle Einträge summiert.

Unterschied der Rahmenfläche Die Fläche des umschließenden Rahmens lässt sich berechnen, indem dessen Breite und Höhe multipliziert werden. Danach erfolgt eine einfache Differenzbildung.

Unterschied des Seitenverhältnisses des Rahmens Das Seitenverhältnis des Rahmens berechnet sich als Verhältnis von Breite zu Höhe. Allerdings lässt sich bei einem ausgerichteten Rechteck nicht eindeutig definieren, welche Seite die Breite und welche die Höhe darstellt. Daher wird zur Berechnung des Verhältnisses die Länge der kürzeren Seite durch die der Längeren dividiert.

Die Merkmale werden wie folgt gewichtet, wobei die einzelnen Werte empirisch bestimmt wurden: der Histogrammdifferenz wird mit $\frac{4}{6}$ das größte Gewicht eingeräumt, während die Ränge aufgrund der beiden anderen Merkmale mit je $\frac{1}{6}$ gewichtet werden.

Abbildung 6: Klassifikation zur Erfassung von Trainingsdaten. Das als positives Beispiel bezeichnete Objekt ist grün markiert.

3.2. Auswahl des Referenzobjekts

Das Referenzobjekt, aufgrund dessen die Berechnung der Differenzmerkmale erfolgt, kann vom Benutzer per Klick auf den angepassten Rahmen in der Anzeige des Bildes ausgewählt werden. Das jeweils als positiv markierte Objekt wird in der grafischen Oberfläche grün markiert, so dass der Benutzer leicht verifizieren kann, dass die Klassifikation aufgrund der von ihm gewählten Referenz korrekt ist (siehe Abbilding 6).

4. Training der Klassifikatoren

In der hier vorgestellten Arbeit werden künstliche neuronale Netzwerke als Klassifikatoren eingesetzt. Es handelt sich hierbei um einfache *Feedforward*-Netze, die in der Literatur bereits hinreichend bekannt sind [7].

4.1. Netzwerkeingabe

Im Allgemeinen sind in der Literatur zwei verschiedene Ansätze zur Klassifikation von Bilddaten mit Hilfe neuronaler Netzwerke bekannt. Zum einen können traditionelle Algorithmen der Bildverarbeitung eingesetzt werden, um zunächst eine Reihe von Merkmalen aus den vorliegenden Daten zu extrahieren. Aus diesen kann anschließend ein Merkmalsvektor konstruiert werden, der die Eingabe für das Netzwerk darstellt. Als Beispiele

können hier die automatische Verifikation von Unterschriften (Bajaj und Chadhury [8]) oder die Gesichtserkennung (Lawrence et al. [9]) aufgeführt werden.

Eine andere Möglichkeit stellt die Verwendung der Bilddaten selbst dar. Hierbei wird das zu klassifizierende Bild meist normiert, so dass die Pixelwerte direkt als Eingabe verwendet werden können. Beispiele hierfür sind unter anderem die optische Zeichenerkennung (LeCun et al. [10]) oder die Klassifizierung von Münzen (Fukumi et al. [11]).

In dieser Arbeit wird der zweite Ansatz gewählt, so dass das Graustufenbild eines segmentierten Objekts als Netzwerkeingabe verwendet wird. Die dort fehlenden Farbinformationen werden in Form eines Histogramms als Eingabe zur Verfügung gestellt. Da durch die Normalisierung des Bildausschnittes Informationen über dessen Größe abhanden kommen, werden zusätzlich einige Eigenschaften des jeweiligen umschließenden Rechtecks genutzt.

Im Folgenden werden die einzelnen Komponenten des Eingabevektors genauer beschrieben.

Rohdaten des Bildausschnitts Der dem Objekt zugehörige Bildausschnitt wird zunächst um den negativen Winkel des umschließenden Rechtecks gedreht, um eine möglichst hohe Invarianz gegenüber Rotation sicherzustellen. Da dies lediglich eine orthogonale Ausrichtung sicherstellt, wird der Bildausschnitt zusätzlich so rotiert, dass die längere Seite des Rechtecks horizontal ausgerichtet ist. Somit wird eine mögliche vertikale Spiegelung des Objekts zwar nicht ausgeschlossen, der üblicherweise länglichen Form eines Rennwagens wird jedoch ausreichend Rechnung getragen.

Anschließend wird der Ausschnitt auf 16 mal 16 Bildpunkte skaliert und in ein Graustufenbild konvertiert. Die durch Division mit 255 normierten Grauwerte jedes Bildpunkts dienen schließlich als Eingangssignale für das Netzwerk.

Histogramm Die Histogrammberechnung erfolgt analog zu der in Abschnitt 3.1.1 beschriebenen Vorgehensweise. Hierbei wird für jede skalierte Häufigkeit ein separater Eingang benötigt.

Form des umschließenden Rechtecks Aus den Eigenschaften des ausgerichteten umschließenden Rechtecks werden mehrere Merkmale extrahiert. Zum einen dient die Fläche des Rechtecks, normiert auf die Maximalgröße eines Objekts (100 auf 100 Bildpunkte), als Eingabe. Daneben werden ebenfalls das Seitenverhältnis und der Winkel, um den das Rechteck gedreht ist, als Eingabe genutzt. Das Seitenverhältnis berechnet sich hier immer als Verhältnis von kürzerer zu längerer Seite. Der Winkel wird durch Division mit 360 normiert.

4.2. Netzwerktopologie

Insgesamt ergeben sich aus den oben beschriebenen Merkmalen 355 Eingabewerte (256 für die Rohdaten, 96 für das Histogramm und 3 für die Eigenschaften des umschließenden Rechtecks). Dahinter befindet sich eine versteckte Schicht mit insgesamt 32 Perzeptronen. Für die Ausgabe wird lediglich ein einzelnes Perzeptron benötigt.

Für die Aktivierungsfunktion der einzelnen Perzeptronen dient eine symmetrische logistische Funktion der Form

$$f(z) = \frac{2}{1 + e^{-z}} - 1$$

Der Wertebereich von $f(z)$ ist $[-1, 1]$, so dass Objekte der vom Netzwerk erlernten Klasse optimalerweise einen Ausgabewert von 1 zur Folge haben.

4.3. Training

Für das Training der Netzwerke wird der gebräuchliche *Backpropagation*-Algorithmus mit *Gradient Decent* eingesetzt, der ursprünglich von Rummelhart et al. vorgeschlagen wurde [12].

Als Voreinstellung für die Lernrate ϵ wird ein Wert von 0.1 gewählt. Anhand zahlreicher Testläufe wurde sichergestellt, dass einerseits ein rasches Lernen gewährleistet ist und gleichzeitig Overfitting so gut wie möglich vermieden wird.

Für die als positiv zu wertenden Trainingsdaten wird eine Ausgabe von 1, für die negativen Beispiele hingegen ein Ausgabewert von -1 angestrebt. Das Training wird als erfolgreich beendet betrachtet, wenn der mittlere quadratische Fehler auf der Trainingsmenge unter 0.001 sinkt.

5. Klassifikation

Im realen Betrieb wird wiederum ein differenzbasierter Ansatz gewählt, um einzelne Objekte zu segmentieren. Analog zu Kapitel 2 werden mittels Differenzbildung gebenüber einem Hintergrundbild Objekte segmentiert, die anschließend mit Hilfe der neuronalen Netzwerke klassifiziert werden.

5.1. Objektklassifikation

Zur endgültigen Klassifikation der aus dem Differenzbild extrahierten Objekte werden die neuronalen Netzwerke verwendet, die hierfür wie in Kapitel 4 konstruiert und trainiert wurden. Da für jede Klasse und damit für jeden Rennwagen ein separater Klassifikator zu Verfügung steht, wird für jedes erkannte Objekt die Ausgabe jedes Netzwerks berechnet, deren Wert sich im Intervall $[-1, 1]$ befindet (siehe Abschnitt 4.2).

Unter den so erhaltenen Ausgabewerten wird nun das Maximum betrachtet. Liegt dieses über 0.9, so wird von einer erfolgreichen Klassifikation ausgegangen. Die dem Objekt letztendlich zugeordnete Klasse ist dabei durch das Netzwerk gegeben, dessen Ausgabe den Maximalwert darstellt. Bei einem maximalen Ausgabewert von unter -0.5 wird das Objekt als Teil des Hintergrundes klassifiziert. Dies ist für die im nächsten Abschnitt beschriebene iterative Hintergrundgenerierung von Bedeutung.

Bei der hier beschriebenen Zuordnung der Objektklassen wird davon ausgegangen, dass alle sich auf der Bahn befindlichen Rennwagen unterschiedliche Klassen repräsen-

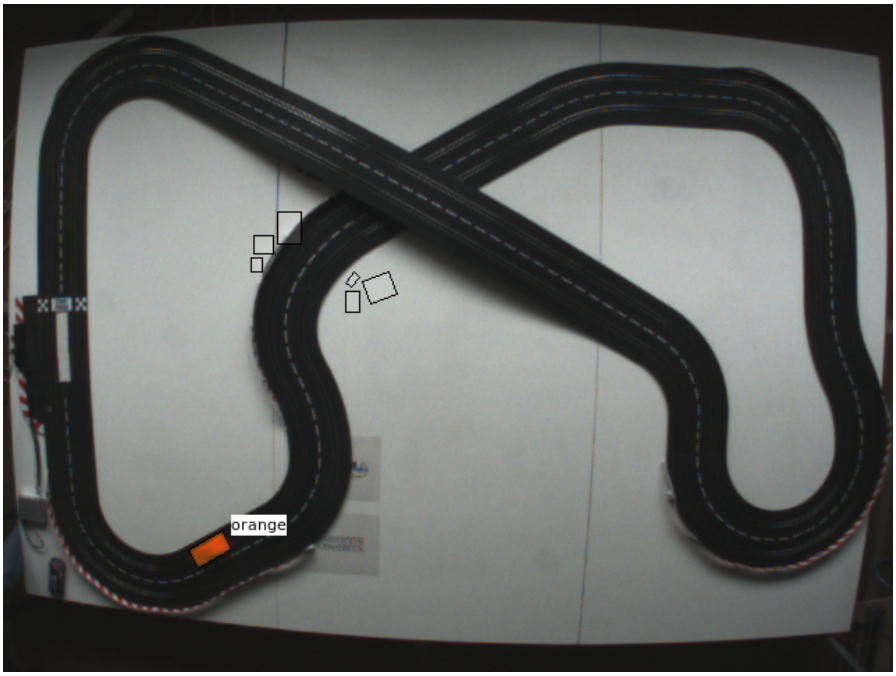

Abbildung 7: Klassifikation von Objekten: Der orangefarbene Rennwagen wurde korrekt erkannt.

tieren. Daher wird, falls mehreren Objekte die gleiche Klasse zugeordnet wird, lediglich das Objekt mit dem maximalen Ausgabewert als Vertreter dieser Klasse betrachtet.

Abbildung 7 veranschaulicht die Ergebnisse der Klassifikation im laufenden Betrieb. Das als Rennwagen erkannte Objekt wurde mit der zugehörigen Klasse beschriftet. Die in der Bildmitte markierten Objekte wurden hingegen als Teile des Hintergrunds klassifiziert.

5.2. Iterative Hintergrundgenerierung

Ein hauptsächliches Problem der oben beschriebenen Klassifikation im realen Betrieb stellt die Bereitstellung eines Hintergrundbildes dar. Ein statischer Hintergrund eignet sich nur bedingt für den Einsatz über eine längere Zeit: bereits eine leichte Änderung der Lichtverhältnisse, oder das Erscheinen zusätzlicher Objekte im Sichtfeld der Kamera können die in Abschnitt 2.1 beschriebene Generierung des Differenzbildes beeinträchtigen. Eine robuste Segmentierung, und damit auch eine akkurate Klassifikation, kann auf diese Weise kaum bewerkstelligt werden.

Es liegt also nahe, das Hintergrundbild dynamisch an den realen Hintergrund der Szene anzupassen. In der Literatur sind hierfür bereits zahlreiche Methoden zur Berechnung eines dynamischen Hintergrundbildes bekannt [13]. Meist dienen statistische Modelle als Grundlage, wie beispielsweise in einem von Ridder et al. entworfenen System, dass jeden Bildpunkt durch einen Kalman-Filter modelliert [14]. Ansätze dieser Art sind für die

hier vorgestellte Arbeit jedoch aufgrund der teilweise hohen Platz- und Zeitkomplexität ungeeignet.

Eine weitere Methode wurde von Heikkilä und Silvén entworfen [15]. Hier wird der Hintergrund fortlaufend gemittelt, während der Vordergrund durch einfache Differenzbildung anhand eines Grenzwertes extrahiert wird (vgl. Abschnitt 2.1). Die Information über den Vordergrund wird nun ebenfalls zur Aktualisierung des Hintergrunds genutzt, indem die einzelnen Bildpunkte über mehrere Kamerabilder hinweg betrachtet werden. Bildpunkte werden nun vom Vordergrund ausgeschlossen und zum Hintergrundbild hinzugefügt, falls sie sich entweder bereits längere Zeit im Vordergrund befinden oder aber häufig zwischen Vorder- und Hintergrund wechseln. So sollen einerseits plötzliche Wechsel der Lichtverhältnisse ausgeglichen und andererseits zufällige Bewegungen im Hintergrund ignoriert werden.

Die hier neu entworfene Methode arbeitet auf ähnliche Weise. Im Gegensatz zu Heikkilä und Silvén wird hier jedoch die in Abschnitt 5.1 beschriebene Klassifizierung zur Aktualisierung des Hintergrundbildes genutzt.

Hierzu wird das Hintergrundbild B also zunächst über die von der Kamera empfangenen Bilder gemittelt. Die segmentierten Objekte werden von der Berechnung jedoch ausgeschlossen, sofern sie nicht als Teil des Hintergrundes klassifiziert wurden (s.o.). Die durch die umschließenden Rechtecke beschriebenen Flächen des Hintegrundbildes werden also nicht aktualisiert und können somit in der nächsten Iteration, also bei Bearbeitung des nächsten Bildes, wiederum segmentiert werden. Auf diese Weise werden insbesondere die als Hintergrund bzw. nicht als Rennwagen klassifizierten Bereiche nach und nach Teil des Hintergrundbildes.

6. Ergebnisse

6.1. Differenzbasierte Segmentierung

Die Verwendung eines Differenzbildes erwies sich in den zahlreichen Testläufen als eine durchaus robuste Segmentierungstechnik. Dies wurde jedoch erst durch die automatische Anpassung des zur Differenzbildung verwendeten Grenzwertes ermöglicht (siehe Abschnitt 2.1). So hatte eine plötzliche Änderung der Lichtverhältnisse meist zur Folge, dass die Segmentierung über einige Bilder hinweg keine verwertbaren Resultate lieferte. War der Grenzwert schließlich ausreichend gut angepasst, konnte mit der Erfassung der Trainingsdaten bzw. der Klassifikation von Objekten fortgefahren werden.

6.2. Automatisierte Erfassung von Trainingsdaten

Die Klassifikation anhand eines Referenzobjekts, wie in Abschnitt 3.1 beschrieben, war in der Praxis durchaus hilfreich. Es ist jedoch festzuhalten, dass die beschriebene Klassifikation keine große Robustheit aufweist. Den größten Störfaktor stellten hierbei Fremdobjekte wie etwa eine am Rand des Bildes auftauchende Hand dar. Da die direkte Umgebung bei der Aufnahme der Trainingsdaten, im Gegensatz zum laufenden Betrieb,

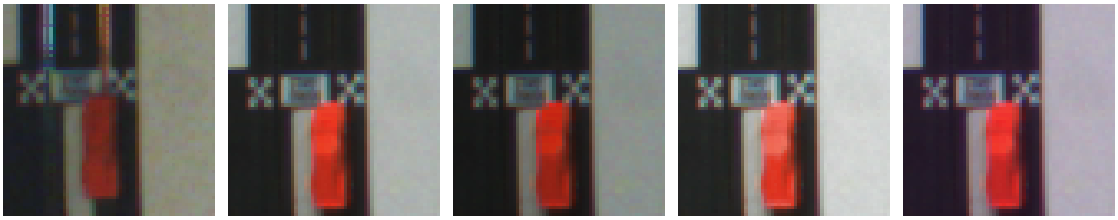

Abbildung 8: Detailaufnahme an der Startlinie. Von links nach rechts: (a) Trainingsdaten; (b) ursprüngliche Testdaten; (c) abgedunkelte und (d) aufgehellte Testdaten; (e) Testdaten mit UV-Verschiebung.

jedoch leicht zu kontrollieren ist, stellt dies im hier behandelten Rahmen kein großes Problem dar.

6.3. Iterative Hintergrundgenerierung

Die in Abschnitt 5.2 beschriebene iterative Generierung des Hintergrundbildes war von großer Bedeutung für die erfolgreiche Segmentierung und Klassifikation im realen Betrieb. Hierdurch wurden Störfaktoren der Segmentierung wie Fremdobjekte oder sich ändernde Lichtverhältnisse zuverlässig ausgeschlossen.

Ähnlich wie die automatische Anpassung des Grenzwertes für die Differenzbildung sorgte auch die Hintergrundgenerierung bei sich schlagartig ändernden Lichtverhältnissen für eine ergebnislose Klassifikation, die über einige Bilder hinweg anhielt.

6.4. Evaluation der Klassifikatoren

Um die Güte der Klassifikation objektiv beurteilen zu können, wurden zuvor aufgezeichnete Bilddaten künstlich verfälscht. Die Verfälschungen wurden in Form von Aufhellung und Abdunklung, aber auch durch die Simulation eines fehlerhaften Weißabgleichs und das Hinzufügen verschiedener Objekte durchgeführt. Eine genaue Beschreibung des Vorgehens zur Erzeugung der Bilddaten befindet sich im Anhang B.

Als Testobjekt diente für diesen Test ein einfarbiger orangefarbener Rennwagen. Die üblicherweise im Motorsport anzutreffenden Werbeaufkleber von Sponsoren, die auch auf zahlreichen Modellen von Carrera zu finden sind, sind auf diesem Wagen nicht vorhanden. Ausgangspunkt der Evaluierung waren zwei aufgezeichnete Bildserien der gleichen Strecke, die jedoch bei unterschiedlichen Lichtverhältnissen aufgenommen wurden. Die erste Bilderstrecke diente zur Erzeugung der Trainingsdaten. Aus der zweiten Bilderstrecke wurden die verfälschten Bilddaten generiert. Somit standen insgesamt 309 Aufnahmen als Trainingsdaten und je 320 Bilder zur Überprüfung zur Verfügung. Abbildung 8 zeigt zum Vergleich den selben Bildausschnitt für jeden der verwendeten Datensätze.

Zur Evaluierung der erlernten Klassifikatoren wurden zunächst die gesamten 320 Einzelbilder der zu testenden Bilderstrecke an das Programm gesendet. Hiermit wurde gewährleistet, dass die Generierung des Hintergrundbildes einen stabilen Zustand erreichen konnte. Anschließend wurden hintereinander die ersten 100 Bilder jeder Serie

Daten	positive Klassifikation (in %)
Ursprüngliche Testdaten	93.8
Abgedunkelt	98.7
Aufgehellt	72.2
UV-Verschiebung	78.6
Fremdobjekt	90.6

Tabelle 1: Ergebnisse der Klassifikation auf verfälschten Bilddaten.

gesendet und für jedes Bild das Ergebnis der Klassifikation festgestellt. Als positives Ergebnis wurde hier die korrekt Erkennung des Rennwagens gewertet. War auf einem Bild kein Rennwagen zu sehen und wurde kein Objekt als Rennwagen klassifiziert, wurde das Ergebnis ebenfalls als positiv interpretiert.

Die Resultate der Evaluation sind in Tabelle 1 aufgelistet. Für die unverfälschten sowie die abgedunkelten Testdaten wurden jeweils 6 Testdurchläufe durchgeführt. Aufgrund der recht starken Varianz der Ergebnisse der Klassifikation wurden mit den aufgehellten und UV-verschobenen Bilddaten je 20 Testläufe durchgeführt. Somit liegen die Ergebnisse für diese beiden Datensätze in einem 95%−Konfidenzintervall von $\pm 4\%$, während dieses für die anderen drei Mittelwerte unter $\pm 1\%$ liegt. Für jeden Testlauf wurde der entsprechende Klassifikator wie in Kapitel 5 beschrieben neu trainiert.

Für die Bewertung der Ergebnisse ist anzumerken, dass eine positive Klassifikation in jedem Bild, also ein Ergebnis von 100%, nicht zwingend notwendig ist. Das *NeuroRacer*-System kann einen Rennwagen auch erfolgreich steuern, wenn für die Dauer einiger Zehntelsekunden keine Informationen über dessen Position vorliegen. Die als negativ gewerteten Klassifikationen der Testläufe stellen zudem in nahezu allen Fällen eine ergebnislose Klassifikation dar. Im realen Betrieb würden also keine falschen Informationen an *NeuroRacer* weitergegeben werden.

Die aufgelisteten Ergebnisse der Klassifikation auf den verfälschten Daten sind unter Beachtung der Trainingsdaten konsistent. Diese waren deutlich dunkler als die Testbilder, welche als Basis für die verfälschten Bilder dienten. Somit ist es schlüssig, dass für die abgedunkelten Testdaten die höchste Rate an positive Klassifikationen erzielt werden konnte, während die Erkennungsrate auf den aufgehellten Bildern am geringsten war. Zur Überprüfung dieser wurden analog dazu die Trainingsdaten verfälscht und die zuvor verwendeten Testdaten für das Training der neuronalen Netzwerke genutzt. Auf den so aufgehellten Bildern wurde zu ca. 94% eine positive Klassifizierung erreicht. Von den abgedunkelten Bilder wurden lediglich noch 48% positiv klassifiziert.

6.5. Laufzeit der Klassifikation

In verschiedenen Tests konnte die Klassifikation um die 50 Bilder pro Sekunde verarbeiten. Der Testrechner besaß einen Intel Core2 Prozessor (T7600, 2.33 Ghz), ein Modell, dessen Alter zum Zeitpunkt der Testläufe bereits über 3 Jahre betrug[3].

[3] `http://ark.intel.com/Product.aspx?id=27257&processor=T7600`

Der Großteil der Zeit, die für die Klassifikation benötigt wird, entfällt auf die Differenzbildung und die iterative Hintergrundberechnung. Die einfachen Bildoperationen wurden daher bereits auf Multiprozessorarchitekturen optimiert und werden automatisch parallelisiert. Die Laufzeit kann somit durch Verwendung von inzwischen üblichen 4- oder 8-Kernprozessoren also nochmals erheblich gesteigert werden. Dies wurde in einem Test mit einem aktuellen 4-Kernprozessor belegt: auf einem Intel Core i7-920 mit 2.67 Ghz wurde eine stabile Verarbeitungsrate von über 60 Bildern pro Sekunde erreicht.

6.6. Vergleich mit Carrera Vision

Die in Abschnitt 1.4 beschriebene Farbsegmentierung von *Carrera Vision* wurde ebenfalls auf den künstlich verfälschten Bilddaten getestet. Die Klassifizierungsfunktion s wurde hierbei auf Grundlage der oben verwendeten Trainingsdaten justiert und anschließend auf den verfälschten Bilddaten getestet.

Die Ergebnisse waren hierbei stark von der manuell festgelegten Funktion s abhängig. Wurde ein relativ breiter Bereich für die Helligkeit der Bildpunkte in der Klasse des Rennwagens gewählt, so war die Segmentierung kaum anfällig für Veränderungen der Helligkeit der Bilder. Folglich führte ein enger Helligkeitsbereich dazu, dass sowohl auf den abgedunkelten als auch auf den aufgehellten Bildern der Rennwagen nur noch in Einzelfällen erkannt wurde.

Auf den mittels UV-Verschiebung verfälschten Bildern brach die Leistung der Farbsegmentierung in beiden Varianten der Funktion stark ein, so dass der Rennwagen nicht mehr klassifiziert werden konnte. Die ist jedoch nicht weiter verwunderlich: Eine Verschiebung der UV-Ebene sorgt für ein farbstichiges Bild. Farbbasierte Segmentierungsmethoden sind daher meist von einem erfolgreichen Weißabgleich abhängig.

Ein zusätzliches Objekt in Wagenfarbe stellte hingegen keine Problem für den in *Carrera Vision* verwendeten Algorithmus dar. Es sei hier jedoch angemerkt, dass im Vorfeld eine manuelle Auswahl des zu verfolgenden Objekts obligatorisch ist.

7. Experimente

Um eine tiefere Erkenntnis über die Relevanz der einzelnen Eingaben der neuronalen Netzwerke sowie die benutzten Trainingsdaten zu erlangen, wurden nun verschiedene Experimente durchgeführt. Die Versuche orientieren sich hier stark an der in Abschnitt 6.4 beschrieben Evaluierung. Als Testdaten dienten daher ebenfalls die künstlich verfälschten Bilder. Die jeweiligen Konfidenzintervalle der hier aufgelisteten Ergebnisse sind jedoch bisweilen um ein vielfaches höher als zuvor, was unter anderem darin begründet liegt, dass weniger Testläufe durchgeführt wurden. Ungeachtet dessen lassen sich einige generelle Tendenzen erkennen.

7.1. Einfluss der Trainingsdaten

Die Relevanz der verwendeten Trainingsdaten wurde untersucht, in dem auf beiden unverfälschten Datensätzen aus Abschnitt 6.4 trainiert wurde. Hierdurch wurde auf fast

allen verfälschten Datensätzen in über 97% der Fälle eine positive Klassifizierung erreicht (Tabelle 2). Hier sei noch angemerkt, dass, aufgrund der von der Umgebung abhängigen Segmentierung, das neuronale Netzwerk durchaus auch Bilder klassifizieren muss, die nicht in den Trainingsdaten enthalten sind.

Datensatz 1	positive Klassifikation (%)
Abgedunkelt	89.3
Aufgehellt	99.0
Datensatz 2	**positive Klassifikation (%)**
Abgedunkelt	99.0
Aufgehellt	98.3
UV-Verschiebung	97.8

Tabelle 2: Ergebnisse der Klassifikation nach Training auf beiden Originaldatensätzen.

7.2. Einfluss der Netzwerkeingabe

Die Relevanz der Eingaben für die Netzwerke wurde, wie die Leistung der Klassifikation selbst, auf verfälschten Testdaten untersucht. Hier wurde die Eingabe der Klassifikatoren jedoch systematisch manipuliert, indem Informationen hinzugefügt oder vorenthalten wurden. Dies betraf sowohl das Training der neuronalen Netzwerke als auch die eigentliche Klassifizierung von Objekten.

Es wurden zunächst verschiedene Kombinationen der Eingaben, wie sie in Abschnitt 4.1 beschrieben werden verwendet. Weiterhin wurde das Graustufenbild in einigen Testfällen durch ein Farbbild ersetzt, so dass die Netzwerke pro Bildpunkt nun 3 Eingänge benötigten. Als Eingabe diente hier der normierte Rot-, Grün- und Blauwert jedes Bildpunkts.

Tabelle 3 listet die Ergebnisse dieser Experimente auf. Zusätzlich sind die Kombinationen mit dem jeweils besten Ergebnis pro Datensatz angegeben (Zeile “Maximum“). Im Allgemeinen lieferten Kombinationen, die statt des Graustufenbildes ein Farbbild verwendeten, die besten Ergebnisse. Auf dem Datensatz mit zusätzlichem Fremdobjekt versagten die so trainierten Klassifikatoren jedoch völlig, so dass fälschlicherweise stets der eingefügte orangefarbene Kreis als Rennwagen identifiziert wurde. Unter anderem ist auch deutlich erkennbar, dass hauptsächlich der auf dem Histogramm basierende Klassifikator (“Histogramm & Form“) auf den aufgehellten und UV-verschobenen Bildern keine zuverlässigen Resultate lieferte.

Abschließend kann festgehalten werden, dass die für die Implementation dieser Arbeit eingesetzte Kombination aus Graustufenbild, Histogramm und Form im Durchschnitt die besten Ergebnisse lieferte. Einige Kombinationen lieferten auf den aufgehellten und UV-verschobenen Bildern bessere Ergebnisse bei der Klassifikation, ließen sich aber durch ein ähnliches Objekt täuschen. Dieser Fall kommt im realen Betrieb jedoch recht häufig vor, so dass sich diese Klassifikatoren nur bedingt für den dortigen Einsatz eignen.

Netzwerkeingabe	Abgedunkelt	Aufgehellt	UV-Versch.	Fremdobjekt
Vollständig	98.7	72.2	78.6	90.6
Bild	84.0	65.0	79.2	78.0
Farbbild	98.0	97.7	98.3	0.0
Bild & Histogramm	98.8	68.5	63.5	91.5
Bild & Form	88.8	72.5	83.8	60.3
Farbbild & Form	98.0	98.0	98.0	0.0
Histogramm & Form	99.0	42.8	56.3	91.0
Vollst. mit Farbbild	98.0	98.0	98.0	0.0
Maximum	Hist. & Form	Farbbild & Form Vollst. Farbbild	Farbbild	Bild & Hist.

Tabelle 3: Ergebnisse der Klassifikation für unterschiedliche Netzwerkeingaben (Angaben in Prozent).

8. Schlussfolgerungen

Zunächst ist zu überprüfen, ob die in Abschnitt 1.5 gesteckten Ziele tatsächlich erreicht wurden. Dies kann im Grunde positiv beantwortet werden. Durch Kombination von bereits bewährten Techniken und einem neuen Ansatz zur Anpassung des in der Segmentierung verwendeten Hintergrundbildes konnte ein System geschaffen werden, dass nahezu selbstständig Rennwagen auf einer Carrerabahn erkennen kann. Die notwendige Benutzerinteraktion beschränkt sich hierbei auf die Auswahl eines geeigneten Referenzobjekts zur Beschaffung der Trainingsdaten.

An zahlreichen Punkten wurde Vorwissen zur Verbesserung der Leistung eingesetzt, beispielsweise bei den Definition von Grenzwerten für die Größe der segmentierten Objekte oder der Verwendung von ausgerichteten umschließenden Rechtecken. An anderen Stellen mussten dagegen experimentell ermittelte Werte verwendet werden, wie etwa für die Schranken des bei der Differenzbildung eingesetzten Grenzwertes. Es wurde somit kein allgemein einsetzbares System entwickelt, sondern vielmehr eine durch den hohen Grad der Automatisierung intelligente Lösung für die Zusammenarbeit mit *NeuroRacer*.

In den durchgeführten Experimenten konnten die Grenzen von traditionellen, farbbasierten Segmentierungsalgorithmen deutlich gemacht werden. Weiterhin wurde gezeigt, dass die hier eingesetzte Kombination von differenzbasierter Segmentierung und maschinellem Lernen zur Klassifikation eine robuste und effiziente Erkennung der Rennwagen erlaubt. Die Anforderungen an die Laufzeit konnten auf handelsüblichen Computern erfüllt werden.

8.1. Einsatz in Carrera Vision

Ein langfristiges Ziel dieser Arbeit stellt die Verwendung des entwickelten Systems in *Carrera Vision* dar. Hier kann das erstmalige Abfahren der Rennstrecke, dass ohnehin zur späteren Positionsbestimmung des Rennwagens nötig ist, etwa zum Erfassen von

Trainingsdaten genutzt werden. Danach kann ein entsprechender Klassifikator automatisch erlernt werden, der dann im weiteren Betrieb verwendet wird. Die zum Abfahren der Strecke notwendige Klassifizierung kann beispielsweise wie in dieser Arbeit anhand eines vom Benutzer gewählten Referenzobjekts erfolgen.

Der Einsatz einer Wissensdatenbank, in der Klassifikatoren und Trainingsdaten verwaltet werden können (siehe Abschnitt C.2), ist hier ebenfalls hilfreich. Einerseits kann der Nutzer etwa bei der Auswahl des Referenzobjekts unterstützt werden. Ebenso ist es denkbar, dass die Klasse des Rennwagens als zusätzliche Information an *NeuroRacer* gesendet wird und so eine eventuell zuvor erlernte Strategie zur Steuerung verwendet oder verbessert werden kann.

8.2. Weitere Anwendungsszenarien

Neben der Erfassung der Position von Rennwagen auf einer Carrerabahn sind auch andere Einsatzmöglichkeiten für das hier vorgestellte Zusammenspiel von Segmentierung und Klassifikation denkbar. Voraussetzung ist jedoch zunächst eine statische Kamera, wie sie beispielsweise auch bei Tischfussballrobotern oder in der *RoboCup* Small-Size League eingesetzt wird [16]. Grundsätzlich sollten sich so die meisten auch vom Menschen leicht erkennbaren Objekte zuverlässig erkennen lassen. Die für die Segmentierung definierten Grenzwerte für die Größe eines Objekts müssten allerdings unter Umständen angepasst werden.

Für Szenarien, in denen bewegliche Kameras benötigt werden, ist die hier vorgestellte iterative Generierung des Hintergrunds nur bedingt geeignet. Grund hierfür ist die für solche Anwendungen zu lange Adaptionszeit an einen völlig neuen Bildhintergrund.

8.3. Mögliche Verbesserungen

Das hier vorgestellte System bietet noch zahlreiche Möglichkeiten für Verbesserungen, die für einen tatsächlichen Einsatz in *Carrera Vision* in Erwägung gezogen werden sollten. Beispielsweise könnte die Robustheit der Differenzbildung weiter erhöht werden, indem statt eines einzelnen Grenzwertes für das gesamte Bild mehrere Grenzwerte für je einen einzelnen Teilbereich des Bildes verwendet werden würden. Hierdurch könnte der störende Einfluss von Fremdobjekten auf die Berechnung des Differenzbildes verringert werden.

Zur Klassifikation selbst wurden in dieser Arbeit künstliche neuronale Netzwerke verwendet. Diese könnten allerdings durch andere geeignete Verfahren ersetzt werden, was unter Umständen die Qualität der Klassifikation weiter erhöhen könnte. Als mögliche Alternative können hier die in den letzten Jahren beliebten *Support Vektor Machines* aufgeführt werden, die bereits erfolgreich zur Bildklassifikation genutzt wurden [17]. Die Verwendung von Techniken wie *Case-Based Reasoning* ist ebenfalls möglich; hier könnten die Trainingsbeispiele zudem im laufenden Betrieb ergänzt werden [18].

Schlussendlich wäre eine Beschränkung der betrachteten Objekte auf den eigentlichen Streckenbereich eine sinnvolle Verbesserung. Hierzu ist allerdings die Anpassungen von einigen der verwendeten Algorithmen sowie das Bereitstellen von entsprechenden Infor-

mationen über die Rennstrecke nötig. Dies ist daher insbesondere für den Einsatz in *Carrera Vision* interessant, da der Streckenverlauf hier ohnehin aufgezeichnet wird.

A. Berechnung der ausgerichteten umschließenden Rechtecke

Wie in Abschnitt 2.3 beschrieben, wird zur Berechnung der umschließenden Rechtecke ein Brute-Force-Algorithmus verwendet. Hierbei wird die zu testende Punktmenge für eine Anzahl von zu testenden Winkeln gedreht. Nach jeder Rotation wird ein an den Hauptachsen ausgerichtetes Rechteck bestimmt, dessen Berechnung vergleichsweise trivial ist. Auf diese Weise wird nun das Rechteck mit minimalem Flächeninhalt gesucht.

Beschreibung des Algorithmus Zunächst wird also das Zentrum der zu betrachtenden Punktmenge P berechnet. Dies geschieht wie bei Gottschalk et al. durch einfache Durchschnittsbildung [5]:

$$\begin{pmatrix} c_x \\ c_y \end{pmatrix} = \frac{1}{|P|} \sum_{p \in P} \begin{pmatrix} p_x \\ p_y \end{pmatrix}$$

Anschließend werden alle Punkte in P für jeden zu testenden Winkel um das berechnete Zentrum c gedreht. Getestet werden Winkel im Bereich $0 \leq \alpha < \pi$, wobei der Abstand der einzelnen Winkel durch die oben genannte Schrittweite gegeben ist.

Die sich so ergebende neue Position eines Punktes p kann durch Multiplikation mit einer entsprechenden Abbildungsmatrix berechnet werden [19]:

$$\begin{pmatrix} p'_x \\ p'_y \\ 1 \end{pmatrix} = \begin{pmatrix} p_x \\ p_y \\ 1 \end{pmatrix} \cdot \begin{pmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ c_x & c_y & 1 \end{pmatrix}$$

α bezeichnet hier den Winkel, um den gedreht werden soll. Die Menge der so berechneten Punkte wird im Folgenden mit P' bezeichnet. Sind alle neuen Koordinaten der Punkte P berechnet, können die Koordinaten des umschließenden Rechtecks R' bestimmt werden:

$$\begin{aligned} r'_{1,x} &= \min_{p' \in P'} p'_x \\ r'_{1,y} &= \min_{p' \in P'} p'_y \\ r'_{2,x} &= \max_{p' \in P'} p'_x \\ r'_{2,y} &= \max_{p' \in P'} p'_y \end{aligned}$$

Die Fläche des Rechtecks berechnet sich also nun zu

$$A = (r'_{2,x} - r'_{1,x}) \cdot (r'_{2,y} - r'_{1,y})$$

Ist der Betrage der Fläche minimal unter den bisher berechneten Flächen, werden die Eckpunkte des Rechtecks, die sich aus r'_1 und r'_2 ergeben, mit der invertierten Abbildungsmatrix für α multipliziert [19]:

$$\begin{pmatrix} r_{1,x} \\ r_{1,y} \\ 1 \end{pmatrix} = \begin{pmatrix} r'_{1,x} \\ r'_{1,y} \\ 1 \end{pmatrix} \cdot \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ -c_x\cos(\alpha) - c_y\sin(\alpha) & c_x\sin(\alpha) - c_y\cos(\alpha) & 1 \end{pmatrix}$$

Die Punkte $\begin{pmatrix} r_{1,x} \\ r_{2,y} \end{pmatrix}$, $\begin{pmatrix} r_{2,x} \\ r_{2,y} \end{pmatrix}$ und $\begin{pmatrix} r_{2,x} \\ r_{1,y} \end{pmatrix}$ werden analog berechnet. Dies sind nun die Eckpunkte des ausgerichteten umschließenden Rechtecks.

Die Anzahl der zu testenden Winkel wurde hier auf 19 beschränkt, die Schrittweite beträgt also 0.1745 (10°). Mittels Optimierungstechniken wie vorberechneten Tabellen für die verwendeten Sinus- und Kosinuswerte konnte auf einem aktuellen PC (Intel Core i7-920, 2.67 Ghz) eine Laufzeit von 2 ms für eine Punktmenge mit $|P| = 10000$ erreicht werden. Die Anzahl der Punkte entspricht hierbei einem voll ausgefüllten Objekt mit maximaler Größe (100 auf 100 Bildpunkte). Die im realen Betrieb auftretenden Punktmengen sind jedoch meist um einiges kleiner.

B. Künstliche Bildverfälschung

Die Verfälschung der Bilddaten für die Evaluierung der erlernten Klassifikatoren (siehe Abschnitt 6.4) wurden mit einer Reihe einfacher Funktionen realisiert.

B.1. Aufhellung

Zur Aufhellung der Bilddaten wird ein RGB-Bildpunkt zunächst in den HSL-Farbraum transformiert. Ein entsprechender Algorithmus hierzu wurde beispielsweise von Agoston formuliert [20]. Mittels Manipulation des l-Wert eines (h, s, l)-Tupels kann nun die Helligkeit des Punktes reguliert werden. Zur Aufhellung wurde die Helligkeit um 20 % erhöht; das in den RGB-Farbraum zurückzurechnende Tupel ist also durch $(h, s, 1.2 \cdot l)$ gegeben.

B.2. Abdunklung

Die Abdunklung eines Bildes wurde analog zur Aufhellung durchgeführt. Hier wird der Helligkeitswert l mit 0.8 statt 1.2 multipliziert. Somit wurde eine Abdunklung um 20 % erreicht.

B.3. UV-Verschiebung

Die UV-Verschiebung soll einen fehlerhaften Weißabgleich simulieren und wird im YUV-Farbraum berechnet. Durch Verschiebung der UV-Ebene erhält das Bild einen Farbstich, da die Farbe Weiß bei korrektem Weißabgleich im Ursprung der Ebene zu finden ist. In Digitalkameras wird üblicherweise bereits ein automatischer Weißabgleich durchgeführt.

Formeln zur Konvertierung zwischen dem RGB- und dem YUV-Farbraum findet man unter anderem bei Pratt [21]. Der U- und V-Wert wurden jeweils um einen Betrag von 5 verschoben, wodurch ein leichter Blaustich entstand.

B.4. Einfügen eines Fremdobjekts

Das Fremdobjekt, das in jedes Bild eingefügt wurde, stellte ein orangefarbener Kreis mit einem Durchmesser von 14 Bildpunkten dar. Die Farbe entsprach in etwa der Farbe des auf der Strecke zu erkennenden Rennwagens. Das Objekt wurde in jedem Bild an einer anderen Stelle platziert, wobei es sich zu Anfang in der linken oberen Ecke befand und dann pro Bild einen Bildpunkt nach unten und einen Bildpunkt nach rechts verschoben wurde.

C. Implementation

Das in der Arbeit entworfene Programm wurde als eigenständiges Softwareprojekt umgesetzt. Es wurde eine graphische Benutzerschnittstelle entworfen, über die das Sammeln von Trainingsdaten, das Trainieren der Netzwerke und die eigentliche Klassifizierung vorgenommen werden können. Für jede dieser Teilaufgaben wurde ein eigenständiger Dialog entworfen, der den Benutzer gegebenenfalls durch die benötigten Einzelschritte führt, wie beispielsweise das Aufnehmen eines Hintergrundbildes.

Das Programm verwaltet Trainingsdaten und Klassifikatoren in einer Wissensdatenbank, die in eine Datei geschrieben bzw. aus einer Datei geladen werden kann. Auf dieser Weise kann über die Zeit eine größere Trainingsdatenbank aufgebaut werden, mit der robuste Klassifikatoren trainiert werden können.

C.1. Verwendete Technologien

Das Programm wurde in der Programmiersprache C++ implementiert.

Die graphische Benutzeroberfläche ist fester Bestandteil des Programms und wurde mit Hilfe des *Qt*-Frameworks[4] realisiert. Die hier vorgestellten Algorithmen und der für die Oberfläche verantwortliche Code sind jedoch logisch voneinander getrennt, so dass die Integration in weitere graphische Umgebungen mit geringem Aufwand möglich ist.

Zur Kommunikation mit der Digitalkamera via Firewire wurde das *Computer Vision Toolkit* (*CVTK*) von Sascha Lange verwendet [2]. Die darin enthaltenen Containerklassen für Bilddaten wurden ebenfalls eingesetzt.

Die Simulation der neuronalen Netzwerke wurde mit *FANN* (*Fast Artificial Neural Network Library*) realisiert [22].

Die Entwicklung der Anwendung erfolgte unter *GNU/Linux*. Für den Realbetrieb von *NeuroRacer* wird jedoch *Mac OS X* verwendet, so dass das Programm dort ebenfalls getestet wurde. Dank der Plattformunabhängigkeit der verwendeten Bibliotheken sowie des eigentlichen Codes war dies relativ problemlos möglich.

[4] `http://qt.nokia.com/`

C.2. Wissensdatenbank

Kernpunkt der entworfenen Anwendung ist eine Wissensdatenbank in Form einer XML-Datei, in der sowohl Trainingsbeispiele als auch Klassifikatoren gespeichert werden können. Trainingsbeispiele werden in einzelne Klassen unterteilt, wobei die Klasse *_negative* für Negativbeispiele reserviert ist. Anhand dieser Klassen lassen sich auf Wunsch automatisiert Klassifikatoren erstellen, wobei jeder Klasse ein gleichnamiger Klassifikator zugeordnet wird. Für das automatische Training werden die Trainingsdaten der jeweiligen Klasse als positive Beispiele gewertet, während die restlichen Trainingsdaten als Negativbeispiele einfließen.

Die XML-Datei besitzt eine einfache Struktur, die den hier gestellten Anforderungen jedoch vollständig genügt. Sie ist in Listing 1 angegeben. Die eigentlichen Daten liegen hierbei in einem Binärformat vor. Die jeweils verwendete Kodierung ist im übergeordneten Element angegeben. Standardmäßig wird die *Base64*-Kodierung verwendet [23].

Die Netzwerkdaten liegen in dem von *FANN* benutzten Format zur Speicherung eines neuronalen Netzwerkes vor. Die Bilddaten sind im jeweils angegebenen Format gespeichert. Hier wird standardmäßig das *Bitmap*-Format verwendet [24].

C.3. Graphische Benutzerschnittstelle

C.3.1. Hauptfenster

Wie oben bereits erwähnt besteht die Benutzeroberfläche aus einzelnen Dialogfentstern. Diese können über ein zentrales Fenster erreicht werden, indem sich eine Übersicht über die aktuell geöffnete Datenbank befindet (Abb. 9). Des Weiteren kann hier eine Wissensdatenbank geladen oder gespeichert werden.

C.3.2. Akquirierung von Trainingsdaten

Dieser Dialog führt den Benutzer durch die notwendigen Schritte zur Erfassung von Trainingsdaten. Zunächst wird das gemittelte Hintergrundbild ,wie in Abschnitt 2.1 beschrieben, aufgenommen. Nachdem der Benutzer einen Rennwagen auf die Strecke gebracht hat, kann er ihn mit einem Mausklick als Referenzobjekt auswählen. Dieses dient nun, wie in Abschnitt 3.1 beschrieben, zur Klassifikation der extrahierten Objekte. Alle nicht als positiv Klassifizierten Beispiele sowie der Bereich des positiven Beispiels mit den Bilddaten des Hintergrundbildes werden als negative Beispiele gespeichert. Abschließend können die positiven Beispiele unter einem frei wählbaren Namen in der Wissensdatenbank gespeichert werden. Die negativen Trainingsbeispiele werden zur Klasse *_negatives* hinzugefügt.

C.3.3. Training der Netzwerke

Neben der automatischen Generierung von Klassifikatoren ist ein manuelles Training der Netzwerke ebenfalls möglich. Der entsprechende Dialog enthält detaillierte Einstellungsmöglichkeiten und ist daher hauptsächlich für Testzwecke geeignet (Abb. 11).

```
<?xml version="1.0" encoding="UTF-8"?>
<wisdom>
  <!-- Netzwerktopologie -->
  <topology>
    <layer count="768"/>
    ...
  </topology>

  <!-- Klassifikatoren -->
  <classifiers>
    <network class="red" encoding="base64" compressed="true">
      <data>
        <!-- Netzwerkdaten wie angegeben -->
      </data>
    </network>
    ...
  </classifiers>

  <!-- Trainingsbeispiele -->
  <examples>
    <set name="_negative">
      <sample>
        <obbox width="50" height="20" angle="40">
          <points>
            <point x="3" y="4" />
            ...
          </points>
          <oriented_box>
            <point x="3" y="4" />
            <point x="5" y="10" />
            <point x="6" y="11" />
            <point x="3" y="14" />
          </oriented_box>
          <aligned_box x="3" y="4" width="3" height="10" />
        </obbox>
        <image encoding="base64" format="bmp">
          <!-- Bilddaten wie angegeben -->
        </image>
      </sample>
      ...
    </set>
    ...
  </examples>
</wisdom>
```

Listing 1: Struktur der XML-Datei zur Speicherung der Wissensdatenbank mit Beispielwerten.

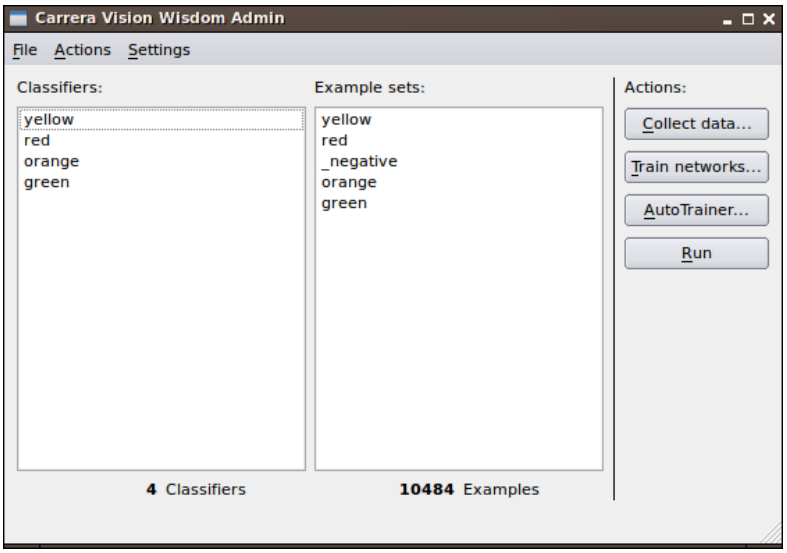

Abbildung 9: Das Hauptfenster der Applikation.

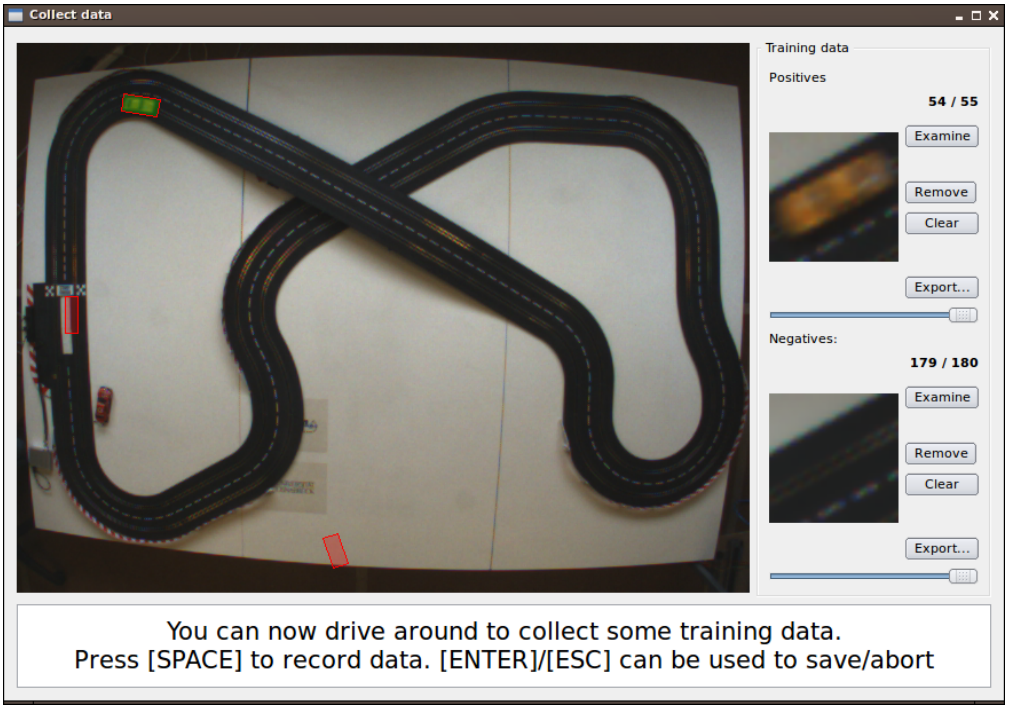

Abbildung 10: Dialog zum Erfassen von Trainingsdaten.

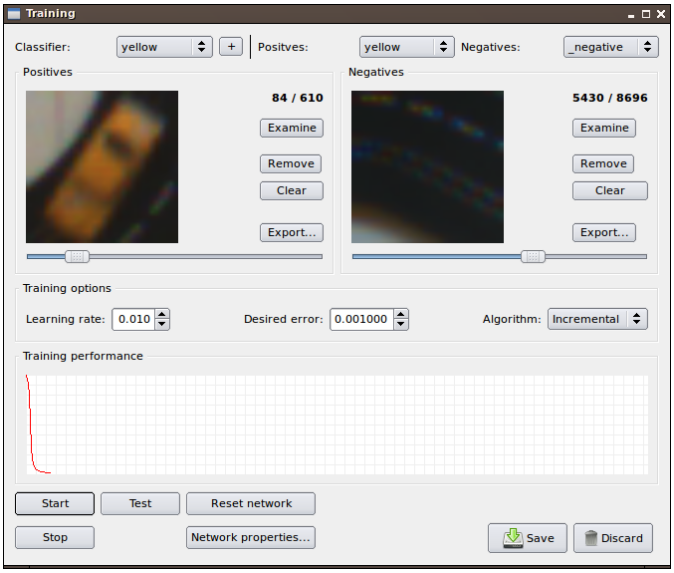

Abbildung 11: Dialog für das Training der Klassifikatoren. Die Kurve zeigt den mittleren quadratischen Fehler an.

C.3.4. Klassifikation

Weiterhin ist ein Dialog verfügbar, der die erlernten Klassifikatoren auf von der Kamera gesendete Bilder anwendet. Das Hintergrundbild für die Differenzbildung wird hier, im Gegensatz zur Akquirierung der Trainingsdaten, iterativ im laufenden Betrieb generiert (siehe Abschnitt 5.2). Somit ist hier keinerlei zusätzliche Benutzerinteraktion nötig.

Literatur

[1] Riedmiller, M., Lange, S., Timmer, S., Hafner, R.: Closed loop simulation system `http://sourceforge.net/projects/clss`.

[2] Lange, S.: Verfolgung von farblich markierten Objekten in 2 Dimensionen. (Oktober 2001)

[3] Egmont-Petersen, M., de Ridder, D., Handels, H.: Image Processing With Neural Networks - a Review (2002)

[4] Dougherty, E.R.: Digital Image Processing Methods. Marcel Dekker, Inc., New York, NY, USA (1994)

[5] Gottschalk, S.: Collision Queries using Oriented Bounding Boxes. PhD thesis (2000)

[6] Lahanas, M., Kemmerer, T., Milickovic, N., Karouzakis, K., Baltas, D., Zamboglou, N.: Optimized Bounding Boxes for Three-Dimensional Treatment Planning in Brachytherapy (2000)

[7] Patterson, D.W.: Artificial Neural Networks: Theory and Applications. Prentice Hall PTR, Upper Saddle River, NJ, USA (1998)

[8] Bajaj, R., Chaudhury, S.: Signature verification using multiple neural classifiers. Pattern Recognition **30**(1) (1997) 1–7

[9] Lawrence, S., Giles, C.L., Tsoi, A.C., Back, A.D.: Face recognition: A convolutional neural network approach. IEEE Transactions on Neural Networks **8** (1997) 98–113

[10] Jackel, L., Battista, M., Baird, H., Ben, J., Bromley, J., Burges, C., Cosatto, E., Denker, J., Graf, H., Katseff, H., LeCun, Y., Nohl, C., Sackinger, E., Shamilian, J., Shoemaker, T., Stenard, C., Strom, I., Ting, R., Wood, T., C., Z.: Neural-net applications in character recognition and document analysis. In: Neural-Net Applications in Telecommunications, Kluwer Academic Publishers (1995)

[11] Fukumi, M., Omatu, S., Takeda, F., Kosada, T.: Rotation-Invariant Neural Pattern Recognition System with Application to Coin Recognition. IEEE Trans. Neural Networks **3**(2) (1992) 272–279

[12] Rumelhart, D.E., McClelland, J.L.: Parallel distributed processing: explorations in the microstructure of cognition, vol. 2: psychological and biological models. MIT Press, Cambridge, MA, USA (1986)

[13] McIvor, A.: Background subtraction techniques (2000)

[14] Ridder, C., Munkelt, O., Kirchner, H.: Adaptive Background Estimation and Foreground Detection using Kalman-Filtering. (1995) 193–199

[15] Heikkilä, J., Silvén, O.: A Real-Time System for Monitoring of Cyclists and Pedestrians. In: VS '99: Proceedings of the Second IEEE Workshop on Visual Surveillance, Washington, DC, USA, IEEE Computer Society (1999) 74

[16] Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E.: RoboCup: The Robot World Cup Initiative (1995)

[17] DeCoste, D., Schölkopf, B., Cristianini, N.: Training invariant support vector machines (2002)

[18] Perner, P., Holt, A., Richter, M.: Image processing in case-based reasoning. Knowl. Eng. Rev. **20**(3) (2005) 311–314

[19] Meyberg, K., Vachenauer, P.: Höhere Mathematik 1. Springer (2001) Sechste, korrigierte Auflage.

[20] Agoston, M.K.: Computer Graphics and Geometric Modelling: Implementation & Algorithms. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2004)

[21] Pratt, W.K.: Digital Image Processing 3rd Edition. Wiley-Interscience (2001)

[22] Nissen, S.: Implementation of a Fast Artificial Neural Network Library (fann). Report, Department of Computer Science University of Copenhagen (DIKU) **31** (2003)

[23] Josefsson, S.: RFC 4648: The Base16, Base32, and Base64 Data Encodings (2006) `http://tools.ietf.org/html/rfc4648`.

[24] Miano, J.: Compressed image file formats: JPEG, PNG, GIF, XBM, BMP. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA (1999)