



Fakultät für Elektrotechnik und Informationstechnik

Professur Prozessautomatisierung

# Bachelorarbeit

**Kamerabasierte Navigation eines Modellfahrzeugs in einem  
Straßenverkehrsszenario**

Daniel Käppler, Leopold Mauersberger

Chemnitz, 22. Oktober 2018

**Prüfer:** Prof. Dr.-Ing. Peter Protzel

**Betreuer:** Dr.-Ing. Sven Lange

**Käppler, Daniel; Mauersberger, Leopold**

Kamerabasierte Navigation eines Modellfahrzeugs in einem Straßenverkehrsszenario

Bachelorarbeit, Fakultät für Elektrotechnik und Informationstechnik

Technische Universität Chemnitz, Oktober 2018

# Inhaltsverzeichnis

<b>Abbildungsverzeichnis</b>	<b>v</b>
<b>Tabellenverzeichnis</b>	<b>viii</b>
<b>1. Glossar, Abkürzungs- und Symbolverzeichnis</b>	<b>ix</b>
Glossar . . . . .	ix
Abkürzungsverzeichnis . . . . .	ix
Generelle mathematische Notationen . . . . .	x
Lateinische Buchstaben . . . . .	x
Griechische Buchstaben . . . . .	xii
<b>2. Einleitung [M]</b>	<b>1</b>
2.1. Hintergrund und Ziel [K] . . . . .	1
2.2. Studentische Wettbewerbe [M] . . . . .	2
2.3. Aufbau der Arbeit [K] . . . . .	4
<b>3. Modelfahrzeug &amp; Testzenario [M]</b>	<b>6</b>
3.1. Hardware [M] . . . . .	6
3.1.1. Fahrzeug . . . . .	6
3.2. Strecke [K] . . . . .	8
<b>4. Grundlagen [K]</b>	<b>11</b>
4.1. Koordinatensysteme [K] . . . . .	11
4.2. Koordinatentransformationen . . . . .	12
4.3. Filter in der Bildverarbeitung [K] . . . . .	14
4.3.1. Gauß-Filter . . . . .	16
4.3.2. Laplace-Filter . . . . .	17
4.4. Kalman-Filter [M] . . . . .	17
4.4.1. Zustandsraumdarstellung . . . . .	17

4.4.2. Filtergleichungen . . . . .	18
4.5. RANSAC <b>M</b> . . . . .	19
4.5.1. Ablauf des Algorithmus . . . . .	19
4.5.2. Nachfolgende Schritte . . . . .	19
4.6. Hough-Transformation <b>K</b> . . . . .	21
4.6.1. Vereinfachte Hough-Transformation . . . . .	22
4.7. Kameramodell <b>M</b> . . . . .	22
4.7.1. Annahmen . . . . .	23
4.7.2. Herleitung . . . . .	24
4.7.3. Entzerrung . . . . .	25
4.7.4. Affine Transformation . . . . .	25
<b>5. Struktur der Software <b>M</b></b>	<b>28</b>
5.1. ROS . . . . .	28
5.1.1. Nodes . . . . .	28
5.2. MATLAB . . . . .	29
5.2.1. Initialisierung . . . . .	30
5.2.2. Callbacks . . . . .	31
5.2.3. Multitasking . . . . .	31
<b>6. Bildvorverarbeitung <b>K</b></b>	<b>33</b>
6.1. Bildentzerrung . . . . .	34
6.1.1. Determination der Auflösung . . . . .	34
6.2. Filterung . . . . .	35
6.3. Binarisierung . . . . .	36
<b>7. Polynombasierte Fahrspurerkennung <b>M</b></b>	<b>37</b>
7.1. Ransac mit Maskierung <b>K</b> . . . . .	38
7.1.1. Approximation der Straßenmarkierungen . . . . .	38
7.1.2. Bestimmung der „Regions of Interest“ . . . . .	39
7.1.3. Initialer Maskenbau . . . . .	40
7.1.4. Vorteile von Ransac mit Maskierung . . . . .	42
7.1.5. Probleme . . . . .	43
7.1.6. Fazit und Lösungsmöglichkeiten . . . . .	44

7.2.	Kalman-Filter <span style="border: 1px solid black; padding: 2px;">M</span>	46
7.2.1.	Fahrspurmodell	47
7.2.2.	Messung	48
7.2.3.	Zustandsraumbeschreibung	50
7.2.4.	Kovarianzmatrizen	52
7.2.5.	konkrete Implementierung des Kalman-Filters	52
7.2.6.	Vorteile	53
7.2.7.	Probleme	53
<b>8.</b>	<b>Punktbasierte Fahrspurerkennung <span style="border: 1px solid black; padding: 2px;">M</span></b>	<b>56</b>
8.1.	Mittellinie <span style="border: 1px solid black; padding: 2px;">K</span>	57
8.2.	Randlinie	57
8.2.1.	Startpunktgewinnung <span style="border: 1px solid black; padding: 2px;">M</span>	58
8.2.2.	zentraler Algorithmus	60
8.2.3.	Sonderfall 1. und 2. Iteration	61
8.2.4.	Ende des Algorithmus	61
8.3.	Verifikation <span style="border: 1px solid black; padding: 2px;">M</span>	63
8.3.1.	Mittellinie <span style="border: 1px solid black; padding: 2px;">K</span>	63
8.3.2.	seitliche Fahrbahnmarkierungen	65
8.4.	Sonderfall unterbrochene Randlinie <span style="border: 1px solid black; padding: 2px;">M</span>	67
<b>9.</b>	<b>Regelung <span style="border: 1px solid black; padding: 2px;">K</span></b>	<b>68</b>
9.1.	Prinzip und Fahrzeugmodell	68
9.2.	Zielpunktgewinnung <span style="border: 1px solid black; padding: 2px;">M</span>	70
9.2.1.	Holen der benötigten Punkte aus der Weltkarte	70
9.2.2.	Kreissegment-Fit	71
9.2.3.	Verschiebung der Kreissegmente	73
9.2.4.	Bildung des Trajektorienkreissegments	73
9.2.5.	Schnittpunktberechnung	75
9.3.	Berechnung des Lenkwinkels <span style="border: 1px solid black; padding: 2px;">K</span>	75
9.4.	Implementierung <span style="border: 1px solid black; padding: 2px;">M</span>	77
<b>10.</b>	<b>Evaluation <span style="border: 1px solid black; padding: 2px;">K</span></b>	<b>79</b>
10.1.	Diskussion prinzipieller Gesichtspunkte <span style="border: 1px solid black; padding: 2px;">M</span>	80
10.2.	Evaluation ohne Ground Truth	82

## **ABBILDUNGSVERZEICHNIS**

---

10.3. Laufzeit [M] . . . . .	83
10.4. Qualität der Weltkarte [K] . . . . .	89
10.5. Geschwindigkeit . . . . .	90
10.6. Hinzufügen von Objekten zum Testszenario [M] . . . . .	94
10.7. Fahren mit weniger Informationen [K] . . . . .	102
10.8. Fazit . . . . .	106
<b>11. Ausblick auf folgende Arbeiten [K]</b>	<b>107</b>
<b>Literatur</b>	<b>108</b>
<b>Anhang</b>	<b>111</b>
<b>A. Daten-DVD</b>	<b>111</b>

# Abbildungsverzeichnis

3.1. Modellfahrzeug „TUCar“ . . . . .	7
3.2. Die Teststrecke mit Kreuzungen, Einbahnstraßen und Parkplätzen . . . . .	8
4.1. Alle eingeführten Koordinatensysteme im Überblick . . . . .	12
4.2. Dreidimensionale Visualisierung von FaltungsfILTERkernen . . . . .	16
4.3. Schema zum Ablauf des RANSAC-Algorithmus . . . . .	20
4.4. Houghtransformation von Geraden: Punkte aus dem Datenraum (a) werden auf den Modellraum (b) abgebildet. . . . .	22
4.5. Das Kameramodell der Omnidirectional Camera Calibration (OCam-Calib)-Toolbox . . . . .	23
4.6. Entzerrung mit dem Kameramodell der OCAMCalib-Toolbox . . . . .	26
4.7. Reales und Ideales Bildsensorkoordinatensystem $\mathcal{S}'$ bzw. $\mathcal{S}$ . . . . .	27
5.1. Struktur des Robot Operating System (ROS)-Systems . . . . .	29
5.2. Struktur des MATLAB-Softwareteils . . . . .	30
6.1. Schritte in der Bildvorverarbeitung . . . . .	33
6.2. verzerrtes Rohbild (a) und entzerrtes Graustufenbild (b) einer Momentaufnahme im Parcours . . . . .	34
6.3. gefiltertes (a) und anschließend binarisertes Bild (b) einer Momentaufnahme im Parcours . . . . .	36
7.1. Approximation der linken (a), mittleren (b) und rechten (c) Fahrbahnmarkierung durch ein Polynom dritten Grades mithilfe jeweiliger RoIs und der Anwendung von RANSAC . . . . .	39
7.2. Der Kern des „Ringfilters“ . . . . .	40
7.3. entzerrtes (a) und binarisiertes Bild (b) einer Testaufnahme auf der Strecke in alten Kameraeinstellungen; Filterergebnis des „Ringfilters“ (c) und Durchschnitt dessen Negation mit dem binarisierten Bild (d) .	41

7.4. Fehlerhaft erzeugte Masken der linken (a), mittleren (b) und rechten (c) Fahrbahnmarkierungen während eines Testlaufes. In Abb. 7.5 ist der Ausschnitt des Geschehens markiert . . . . .	44
7.5. Plot der Weltkarte nach einer Testrunde mittels manueller Steuerung. Die braune Umrandung bezeichnet das in Abbildung 7.4 dargestellte Sichtfeld . . . . .	45
7.6. Plot einer Iteration des Kalmanfilters . . . . .	47
7.7. „Weglaufen“ des Kalmanfilter-Zustands . . . . .	54
8.1. Startpunktdefinition des Riverflow-Algorithmus für die Randlinien anhand der mittleren Fahrbahnmarkierung . . . . .	58
8.2. Startpunktdefinition des Riverflow-Algorithmus für die Randlinien durch eindimensionale Hough-Transformation . . . . .	59
8.3. Funktionsweise des Riverflow-Algorithmus . . . . .	59
8.4. Plot eines Riverflow-Durchlaufs für die Randlinien . . . . .	62
8.5. Detektion von Punktgruppen mit bestimmten Eigenschaften mithilfe der <i>regionprops</i> -Funktion . . . . .	64
9.1. Modellierung der Ackermannlenkung durch übliche Vereinfachung auf das Bicycle-Modell in Anlehnung an (Corke 2017) . . . . .	69
9.2. Ermittlung der Rotationsrichtung eines Kreissegmentes . . . . .	72
9.3. Zielpunktgewinnung . . . . .	74
9.4. Visualisierung zur Bestimmung des Kreisradius aus dem gegebenen Zielpunkt $q$ in Roboterkoordinaten . . . . .	76
10.1. Zeitbedarf aller Fahrspurverfolgungskomponenten bei 5 Hz, 4 Hz, 3 Hz und 1 Hz Bildfrequenz . . . . .	85
10.2. Schwankungen Periodendauer Bildaufnahme/Bild-Callback bei 3 Hz Bildfrequenz . . . . .	86
10.3. Zeitbedarf aller Fahrspurverfolgungskomponenten pro Aufnahme bei 3 Hz Bildfrequenz; Median aller Messwerte einer Runde der Teststrecke	86
10.4. Zeitbedarf aller Fahrspurerkennungskomponenten bei 3 Hz Bildfrequenz	87
10.5. Zeitbedarf der Startpunktfindungsalgorithmen für die Erkennung der seitlichen Fahrbahnmarkierungen bei 3 Hz Bildfrequenz . . . . .	89
10.6. Übertragungszeit der Aufnahmen bei 3 Hz Bildfrequenz . . . . .	90

10.7. Darstellungen aufgenommener Karten des Szenarios zur Qualitätsuntersuchung	91
10.8. Boxplots zur Menge der Punkte, welche in einem Regelungsintervall aus der Weltkarte entnommen und zur Zielpunktberechnung verwendet werden . . . . .	92
10.9. Anteil der unbestimmten Zielpunkte einer Runde auf dem Parcours mit verschiedenen Geschwindigkeiten . . . . .	93
10.10Vorversuch Kontraste zusätzlicher Objekte im Testszenario . . . . .	96
10.11Versuchsaufbau farbige Quader im Testszenario . . . . .	97
10.12Histogramme Hypothesenzahl mit „Störobjekten“ im Testszenario; Variation des Abstandes von der Fahrspur . . . . .	98
10.13Beispiele für die Erkennung mehrerer Hypothesen mit „Störobjekten“ im Testszenario . . . . .	99
10.14Zusammenhang Anzahl verfolgter Hypothesen & Dauer der Detektion seitlicher Fahrbahnmarkierungen; 4 cm Abstand Quader zu Fahrspur .	100
10.15Versuchsaufbau mit „Störobjekt“ auf der Gegenfahrspur . . . . .	100
10.16Versuch mit „Störobjekt“ auf der Gegenfahrspur . . . . .	101
10.17Testfahrt mit durch weißes Papier abgedeckten Mittelstrichen . . . . .	102
10.18Ungünstige Position für die mit Hough initialisierte Randlinienerkennung	103
10.19Testfahrt mit durch weißes Papier abgedeckte Randlinien . . . . .	104
10.20Punkterkennung bei abgedeckten Randlinien . . . . .	105
10.21Demonstration der Schwäche des Riverflow bei größeren Unterbrechungen in der Randlinie . . . . .	105

## **Tabellenverzeichnis**

# 1. Glossar, Abkürzungs- und Symbolverzeichnis

## Glossar

### TUCar

Wortschöpfung für die Bezeichnung des entwickelten Modellfahrzeugs. TUC steht für Technische Universität Chemnitz und geht in das englische Wort für „Auto“ über.

## Abkürzungsverzeichnis

ICR	Instantaneous Centre of Rotation
IMU	Inertial Measurement Unit
KS	Koordinatensystem
LoG	„Laplacian of Gaussian“, oder auch Marr-Hildreth-Operator
OCamCalib	Omnidirectional Camera Calibration
RANSAC	„ <b>R</b> andom <b>A</b> mple <b>C</b> onsensus“, deutsch etwa „Übereinstimmung mit einer zufälligen Stichprobe“
RoI	„ <b>R</b> egion <b>of</b> <b>I</b> nterest“, deutsch: „Bereich von Interesse“

ROS              Robot Operating System

## Generelle mathematische Notationen

$\text{atan2}(y, x)$	erweiterte Umkehrfunktion des Tangens $\tan(\alpha) = \frac{y}{x} \rightarrow \text{atan2}(y, x) = \alpha$ mit Wertebereich $\alpha = 0 \dots 360^\circ$
$ x $	Betrag von $x$
$\tilde{x}$	Eine Tilde über einer Variablen symbolisiert eine korrigierte Größe
$f'$	Ableitung der Funktion $f$
$f' _x$	Ableitung der Funktion $f$ an der Stelle $x$
$\dot{f}$	Ableitung der Funktion $f$ nach der Zeit
$f(x)$	allgemeine Funktion in Abhängigkeit der Variable $x$
$\mathbf{X}^{-1}$	Inverse der Matrix $\mathbf{X}$
$\mathbf{X}$	Fettgedruckte Großbuchstaben bezeichnen eine Matrix
$\max(\mathbf{x})$	Maxima von $\mathbf{x}$
$\bar{x}$	Mittelwert der Variable $x$
$\ \mathbf{x}\ $	Euklidische Norm des Vektors $\mathbf{x} = \begin{pmatrix} x_x \\ x_y \end{pmatrix}: \ \mathbf{x}\  = \sqrt{x_x^2 + x_y^2}$
$\hat{x}$	Ein Circumflex über einer Variablen symbolisiert eine prädizierte Größe
$x$	Kleinbuchstaben bezeichnen einen Skalar
$\mathbf{X}^T$	Transponierte der Matrix $\mathbf{X}$
$\mathbf{x}$	Fettgedruckte Kleinbuchstaben bezeichnen einen Punkt oder Vektor

## Lateinische Buchstaben

$c_0$	Fahrspurkrümmung: Änderungsrate des Gierwinkels
$c_1$	Änderungsrate der Fahrspurkrümmung
$e$	eulersche Zahl $e \approx 2,718$
$w$	Radstand (mittlerer Abstand zwischen Vorder- und Hinterachse des Fahrzeugs)
$m$	Anstieg einer Gerade ( $dy/dx$ )
$n$	Achsenabschnitt einer Gerade ( $y$ -Wert bei $x = 0$ )
$\mathbf{a}$	Abstandsvektor
$r$	Radius eines Kreises
$t$	Zeit
$v$	Geschwindigkeit
$x$	x-Koordinate
$y$	y-Koordinate
$y_{off}$	Lateraler Versatz: Abstand der Fahrspurmitte vom Koordinatenursprung des Linien-Koordinatensystem (KS) $\mathcal{L}$
$z$	z-Koordinate
$\mathbf{m}$	Mittelpunkt eines Kreises
$q$	Zielpunkt für den „Pure-Pursuit“-Regler
$\mathbf{d}$	Verschiebungsvektor
$\mathbf{t}$	Translationsvektor
$\mathbf{u}$	Eingangsvariablenvektor
$\mathbf{x}$	Zustandsvektor
$\mathbf{y}$	Ausgangsvariablenvektor
$\mathbf{A}$	Systemmatrix
$\mathbf{B}$	Eingangsmatrix
$\mathbf{C}$	Ausgangsmatrix/Messmatrix
$\mathbf{D}$	Durchgangsmatrix
$\mathbf{I}$	Einheitsmatrix
$\mathbf{R}$	Rotationsmatrix
$\mathbf{S}$	Skalierungsmatrix
$\mathbf{T}$	homogene Transformationsmatrix
$\mathcal{I}$	Bild-/Imagekoordinatensystem

$\mathcal{L}$	Linienkoordinatensystem
$\mathcal{C}$	Kamera-KS
$\mathcal{S}$	ideales Bildsensor-KS der OCamCalib-Toolbox
$\mathcal{S}'$	reales Bildsensor-KS der OCamCalib-Toolbox
$\mathcal{B}$	Roboter-/Bodykoordinatensystem
$\mathcal{W}$	Weltkoordinatensystem

## Griechische Buchstaben

- $\gamma$  Lenkwinkel: mittlere Winkelstellung der Vorderräder des Autos zur  $x_{\mathcal{B}}$ -Achse
- $\theta$  Orientierung des Roboters als Teil der Pose
- $\pi$  Die Kreiszahl Pi  $\pi \approx 3,14$
- $\sigma$  Standartabweichung
- $\varphi$  Gierwinkel: Winkel zwischen Abszisse des Fahrspurkoordinatensystems und Tangente an das Fahrspurpolynom bei  $x = 0$
- $\rho$  Abstand eines Punktes vom Mittelpunkt des Bildsensors

## 2. Einleitung M

Autonomes Fahren ist schon längst kein Sciencefiction mehr. Prototypen autonomer Fahrzeuge existieren schon seit geraumer Zeit (Kröger 2015). Doch gerade in den letzten Jahren hielten Funktionalitäten autonomer, mobiler Roboter in Form von Fahrerassistenzsystemen Einzug in Serienfahrzeugen. Besonders die Erkennung und Verfolgung von Fahrspuren stellt dabei eine wiederkehrende Aufgabe dar (Kunze u. a. 2018; Narote u. a. 2018) und kann als ein Hauptbestandteil dieser Arbeit angesehen werden.

Um auch Studierenden der Technischen Universität Chemnitz einen Einblick in die Welt der führerlosen Automobile zu bieten, wurde im Kontext dieser Arbeit eine Modellumgebung samt Fahrzeug entwickelt. Die entstandene Hard- und Software soll in kommenden Jahren bei Praktikumsaufgaben zur Vorlesung „Autonome Systeme“, weiteren Lehrveranstaltungen sowie diversen studentischen Arbeiten zum Einsatz kommen.

### 2.1. Hintergrund und Ziel K

Der Anstoß dieses Projektes ist die Idee, in Zukunft einen weiteren Praktikumsversuch aufzubauen. An der Professur Prozessautomatisierung der TU Chemnitz besteht bereits ein Projektpraktikum, in dem mit eigens dafür entwickelten Robotern ein schachbrettartiges Labyrinth autonom durchfahren werden muss. Diese „TUCbots“ besitzen allerlei Sensoren wie Encoder, Infrarot-Abstandsdetektoren und Anschlagschalter, jedoch keine Kamera. Das Ziel ist nun, einen weiterführenden, neuen Praktikumsaufbau zu schaffen, in dem sich ein Modelfahrzeug mit Ackermannlenkung einzig mithilfe einer omnidirektionalen Kamera zurecht finden soll.

Mit diesem Hintergrund bestand unsere Absicht darin, einen Fahrspurverfolgungsalgorithmus zu implementieren, welcher durch mögliche spätere Aufgabenstellungen für das Praktikum erweiterbar sein sollte. Dementsprechend mit dieser Arbeit ver-

bundene Ziele sind folgend genannt:

- Entwurf einer geeigneten Teststrecke
- Entzerren festgelegter Bildausschnitte einer omnidirektionalen Kamera
- Filtern von Bildern
- Linienerkennung in einem verarbeiteten Bild und damit verbundene mathematische Approximation der Fahrbahnmarkierungen
- aktuelle Pose des Autos in Bezug eines globalen Koordinatensystems bestimmen
- Aufbau einer Karte zur Pfadplanung
- Trajektorienverfolgung durch Regelung des Lenkwinkels

Damit das Ergebnis der Ausarbeitung kein bloßes, auf das aktuelle Bild reaktives Fahrverhalten ist, stand das Eintragen von Informationen in eine Weltkarte während des Linienerkennungsprozesses von Anfang an fest.

Die initiale, hauptsächliche Inspirationsquelle zur Themenwahl des vorliegenden Projektes stellten jedoch schon vorhandene, unter studentischen Teilnehmern ausgetragene Turniere mit breitem Aufgabenspektrum im Bereich führerloser Automobile dar, über die im folgenden Abschnitt berichtet werden soll.

## 2.2. Studentische Wettbewerbe M

Um Studierenden möglichst früh einen Einblick in die Welt selbstständig fahrender Automobile zu bieten, wurden in der Vergangenheit eine Vielzahl von Wettbewerben ausgetragen. Wie auch in dieser Arbeit bedient man sich hier oft eines Modelfahrzeugs, um die Kosten überschaubar zu halten und die Testumgebung kontrollierter gestalten zu können. Im Folgenden möchten wir einige dieser Wettbewerbe kurz vorstellen.

### 2.2.0.1. Carolo-Cup

Seit 2008 wird an der Technischen Universität Braunschweig der Carolo-Cup<sup>1</sup> ausgetragen. Neben den sogenannten dynamischen Disziplinen

---

<sup>1</sup><https://wiki.ifr.ing.tu-bs.de/carolocup/carolo-cup>

- Rundkurs ohne Hindernisse
- Rundkurs mit Hindernissen
- Einparken

welche mittels eines durch die Teams entwickelten Fahrzeugs im Maßstab 1:10 bestmöglich bewältigt werden sollen, wird in den statischen Disziplinen

- Präsentation und Konzept
- Technische Ansätze

auf die Soft-Skills der Teammitglieder sowie die Innovation und ökonomische Bilanz beim Bau des Fahrzeugs Wert gelegt. Das Regelwerk des Carolo-Cups gab beim Entwurf der Testumgebung für das in dieser Arbeit verwendete Fahrzeug wichtige Anhaltspunkte, auch unsere Teststrecke bildet einen zweispurigen Rundkurs und kann zum Absolvieren verschiedener Sonderaufgaben wie Einparken, Verhalten an einer Kreuzung und Einbiegen in eine Nebenstraße verwendet werden.

### 2.2.0.2. Audi Autonomous Driving Cup

Auch der Audi Autonomous Driving Cup<sup>2</sup> bot Inspiration für die folgenden Kapitel. Im Gegensatz zum Carolo-Cup wird bei diesem Turnier die Hardware samt eines Software-Frameworks vom Veranstalter gestellt. Somit verlagert sich der Schwerpunkt noch weiter in Richtung der Findung&Implementierung schneller&robuster Algorithmen.

### 2.2.0.3. Weitere Wettbewerbe

Als weitere studentische Wettbewerbe, auf die wir erst im späteren Verlauf der Arbeit aufmerksam wurden, sind zu nennen:

**NXP-Cup** Auch beim NXP-Cup<sup>3</sup> werden weite Teile der Hardware vorgegeben. Die Implementierung der Algorithmen auf einem wenig performanten Mikrocontroller stellt eine besondere Herausforderung dar.

<sup>2</sup><https://www.audi-autonomous-driving-cup.com/>

<sup>3</sup><https://community.nxp.com/groups/tfc-emea>

**AI Driving Olympics** Die Artificial Intelligence Driving Olympics<sup>4</sup> beschäftigen sich mit Künstlicher Intelligenz im Kontext autonomes Fahren, eine Kombination die durch moderne Rechentechnik klassischen Ansätzen oft überlegen ist.

## 2.3. Aufbau der Arbeit K

Der folgende Inhalt der Ausarbeitung ist ähnlich der Entstehungsreihenfolge aufgebaut. Anfangs finden sich neben der Vorstellung der physikalisch greifbaren Betriebsmittel (Kapitel 3) grundlegende Beschreibungen der genutzten Methoden der Bildverarbeitung und Mathematik (Kapitel 4). Die Verarbeitung der Bilder und die Berechnungen für die Steuerung des Autos führen wir auf einem externen Rechner mit MATLAB durch. Wie die Software dazu aufgebaut ist, wird im anschließenden Kapitel 5 erläutert. Der danach folgende Hauptteil befasst sich mit der Bildverarbeitung (Kapitel 6 und 7) und Trajektorienplanung (Kapitel 9). Da wir uns im Vorfeld nicht auf einen speziellen Algorithmus für die Fahrspurverfolgung festgelegt haben und einen möglichst gut funktionierenden Ansatz finden wollten, stellen wir drei bearbeitete Methoden zur Fahrspurerkennung vor (Abschnitte 7.1, 7.2, 8), auch wenn letztlich nur eine Vorgehensweise lauffähig implementiert wurde. Nach einer umfassenden Evaluation (Kapitel 10) blicken wir zum Abschluss auf mögliche Aufgaben, die man zukünftig mit dem Modellfahrzeug in dem Straßenverkehrsszenario bewältigen könnte (Kapitel 11).

### 2.3.0.1. Aufteilung M

Diese Arbeit ist eine Gruppenarbeit. Während der Einarbeitungsphase in grundlegende Methoden der Bildverarbeitung, dem Kerngebiet dieser Arbeit, wurden Problematiken vorrangig zusammen erörtert. Nach dem individuellen Auseinandersetzen mit vielen Dokumenten zu ähnlichen Projekten des Themengebiets „Autonomes Fahren mit Modellfahrzeugen“ ergaben sich viele Einzelkomponenten, die zu einer lauffähigen Fahrspurverfolgung nach unseren Vorgaben notwendig sind. Bei der Implementierung dieser Verfahren beschäftigte sich je ein Autor vorrangig mit dem Sachverhalt. Die verschiedenen Programmbausteine sind dennoch oft durch die Überlegungen beider Gruppenmitglieder entstanden, da Ideen oft gegenseitig opti-

<sup>4</sup><https://www.duckietown.org/research/ai-driving-olympics>

miert und auf Übergabeparameter anderer Programmteile abgestimmt wurden. Nach Abschluss des praktischen Teils der Arbeit verfasste der Ideengeber eines jeden Elements der Software das entsprechende Kapitel. Aufgrund dieses Gruppenarbeitscharakters soll kenntlich gemacht werden, wer Urheber eines jeden Abschnitts ist. Die Markierung erfolgt durch K für Daniel Käppler und M für Leopold Mauersberger. Das zuletzt gegebene Label zeigt den Autor des aktuellen Textes an, bis das jeweils andere Symbol einer Überschrift angefügt ist.

## 3. Modellfahrzeug & Testzenario M

Die präziseste Forschung zum Themengebiet autonomes Fahren kann mit echten PKW im realen Straßenverkehr oder auf einer daran angelehnten Teststrecke stattfinden. Da dies jedoch weder technisch, finanziell oder rechtlich möglich ist und die im Bachelorstudium erworbenen Fähigkeiten oftmals übersteigt, muss ein geeignetes Modell geschaffen werden. Neben der notwendigen Reduktion der Komplexität wurden die signifikantesten Rahmenbedingungen hierbei durch:

- die Größe des Labors, in dem die Testumgebung aufgebaut werden sollte
- die Menge der käuflichen Fahrplattformen mit Ackermannlenkung

vorgegeben.

Weiterhin wurde bewusst auf die Nutzung „einfacher“ Sensoren wie Infrarot-Abstandsdetektoren, Anschlagsschalter oder Ultraschall-Entfernungsmesser verzichtet, da an der Professur schon eine Fahrplattform mit diesen Sensoren existiert und sie wenig Mehrwert zu den Informationen eines Kamerabildes bieten. Die Umbauten am gewählten Modellfahrzeug „TUCar“ sowie die Überlegungen zum Entwurf der Strecke sollen in den anschließenden zwei Abschnitten kurz erläutert werden.

### 3.1. Hardware M

#### 3.1.1. Fahrzeug

Das in dieser Arbeit genutzte Modellfahrzeug wurde von Mitarbeitern der Professur für unsere Untersuchungen und folgende Projekte aufgebaut. Es handelt sich um ein autoähnliches Allrad-Chassis im Maßstab 1:18, an welchem für diese Arbeit notwendige Umbauten ausgeführt wurden (s. Abb. 3.1).



Abbildung 3.1.: Modellfahrzeug „TUCar“

**Motor** Der originale Elektromotor wurde gegen ein Modell mit Encoder ersetzt, da Odometrieinformationen für die implementierte Regelung unerlässlich sind. Zusätzlich ist dem Motor ein Getriebe nachgeschaltet, welches ein Fahren mit langsameren Geschwindigkeiten ermöglicht. Tests mit wenig optimierten Algorithmen werden somit signifikant erleichtert.

**Inertial Measurement Unit (IMU)** Um eine genauere Bestimmung der Pose zu ermöglichen, ist eine IMU zur Ermittlung der axialen sowie radialen Geschwindigkeiten und Beschleunigungen am Fahrzeugs angebracht. Auch die durch numerische Integration erhaltene Orientierung kann direkt von diesem Modul abgefragt werden.

**Kamera** Neben der Odometrie stellt eine in Vogelperspektive angebrachte Kamera mit Fisheye-Objektiv die einzige Sensorik des Fahrzeugs dar.

**Rechentechnik** Zur Ansteuerung der Kamera und zum Entgegennehmen der Fahrbefehle ist ein Raspberry-PI Einplatinenrechner montiert, dessen WLAN-Schnittstelle zum Informationsaustausch mit dem Fahrzeug dient.

### 3.2. Strecke K

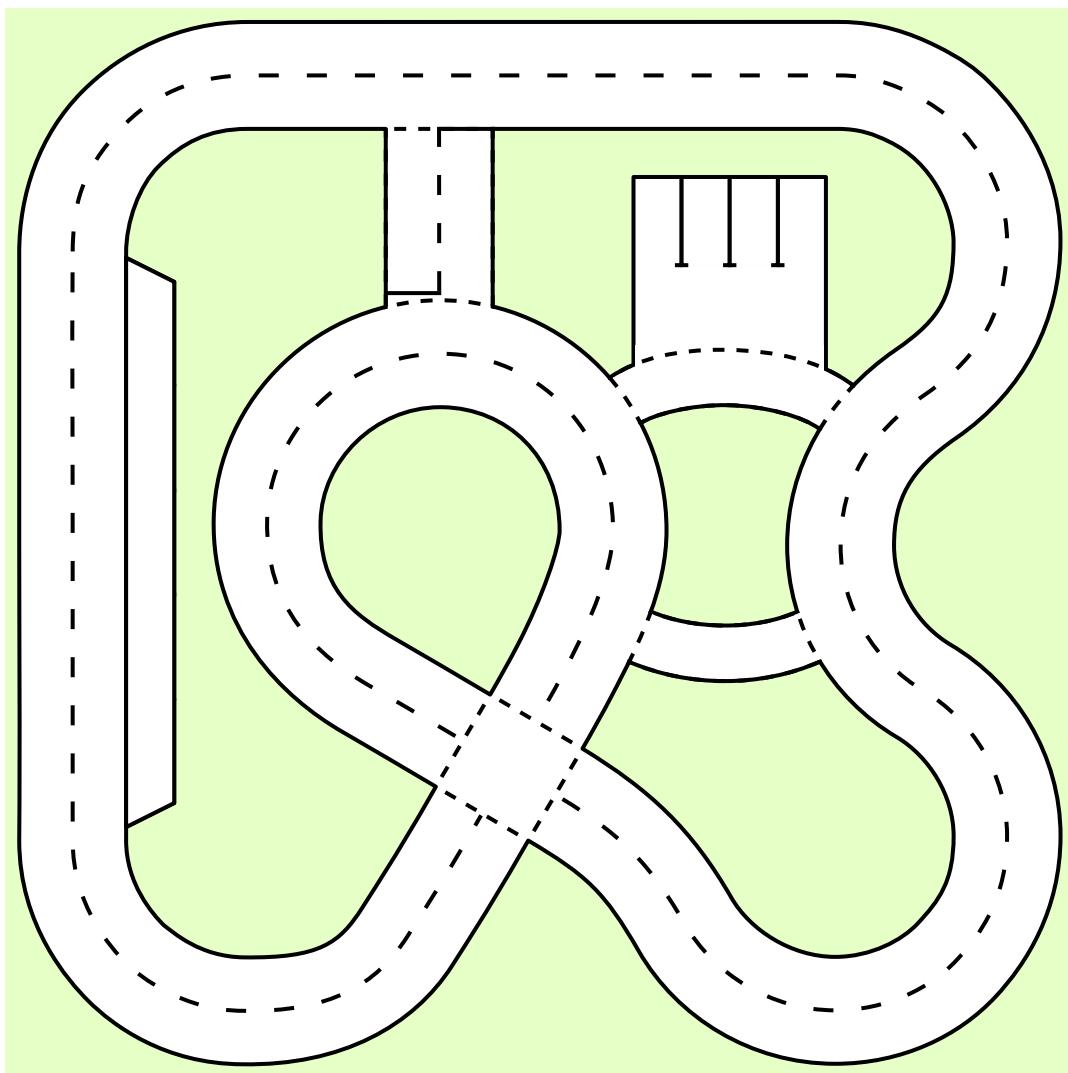


Abbildung 3.2.: Die Teststrecke mit Kreuzungen, Einbahnstraßen und Parkplätzen

Das Straßenverkehrsszenario ist ein Rundkurs, der von sich selbst und weiteren Verbindungsstraßen gekreuzt wird. Er ist dem echten Straßenverkehr ansatzweise nachempfunden. Ein vorrangiges Ziel in der Entwurfsphase war, eine vielfältige, wiederverwend- und erweiterbare Testumgebung zu schaffen, welche später im ange- dachten Praktikumsversuch Anwendung finden kann. Eine gute Orientierung dafür

bot hierbei der Carolo-Cup (siehe Abschnitt 2.2.0.1). Auf extra eingezeichnete Hinweise wie Schilder oder Pfeile wurde bewusst verzichtet, damit der Parcours vielfältig einsetzbar bleibt.

### 3.2.0.1. Randbedingungen

Das Straßenverkehrsszenario ist unter folgenden Maßgaben entworfen worden:

- Das Platzangebot im Labor ist leider stark begrenzt, was auch der Grund für die Wahl des eher kleinen 1:18-Fahrzeugmodells ist. Für die Straßenlandschaft wurde nach längeren Überlegungen eine Abmessung von  $4 \times 4$  Metern vorgegeben.
- Unter dem Gesichtspunkt der perspektivischen Weiterverwendung in einem Praktikumsversuch ist es wünschenswert, den Untergrund möglichst robust und flexibel zu gestalten.
- Die Parameter Straßenbreite und Parkplatzgrößen sollten natürlich dem Auto angepasst sein.
- Damit das Modellfahrzeug ordnungsgemäß durch das Verkehrssystem navigiert werden kann, darf der Radius der Kurven nicht zu klein sein, da das Auto einen maximalen Lenkwinkel von circa  $30^\circ$  besitzt.
- Auf dieser Strecke sollen zukünftig neben einer normalen Fahrspurverfolgung auch Überholen, Abbiegen, Vorfahrt gewähren, Erkennen einer Einbahnstraße und Einparken in Parktaschen verschiedener Ausführung implementiert werden können. Auch wenn bei weitem nicht alles Ziel unserer Arbeit ist, kann man sich diese Operationen als mögliche Aufgaben für den späteren Praktikumsaufbau vorstellen. Somit steht fest, dass sowohl breite Straßen mit Mittellinienstrichen als auch schmale Einbahnstraßen, Kreuzungen und Längs- und Querparkplätze vorgesehen sind.
- Der einzige Umfeld-Sensor des Autos ist dessen Kamera. Daher ist ein starker Kontrast der Fahrbahnmarkierungen zum Hintergrund erwünscht

Gegenüber eines einfachen Rundkurses soll hier durch Zusätze wie Querstraßen, Kreuzungen und Parkplätze die Herausforderung etwas erhöht werden, eine robuste Fahrspurerkennung zu entwickeln.

### 3.2.0.2. Gestaltung

Da der Platz sehr begrenzt ist, wurde darauf geachtet, möglichst viel der zur Verfügung gestellten Fläche mit Straße zu versehen. Zur optischen Hervorhebung der Fahrbahn und der für das Auge visuellen Ansprechbarkeit ist der Untergrund mit einem hellen Grün bedruckt. Die aus den zuvor genannten Bedingungen entstandene Gestaltung der Teststrecke ist in Abbildung 3.2 gezeigt. Das Straßenverkehrsszenario ist auf zwei  $4 \times 2$  Meter große, robuste PVC-Werbeplanen gedruckt worden. Diese Planen können jederzeit aufgerollt und aneinandergelegt werden.

Die Breite einer Fahrbahn wurde anhand des Modellfahrzeugs auf 20 cm festgelegt. Auch die Parkplätze sind auf speziell dieses Auto angepasst und verfügen im Maßstab über straßenverkehrsähnliche Dimensionen.

Der relativ stark beschränkte Lenkwinkel und der Umstand des Platzmangels im Labor ließen nicht übermäßig viele Möglichkeiten in der Kurvenkrümmung offen. Daher besitzen alle Kurven nahezu gleiche Radien. Zur Bestimmung des minimalen Radius ließen wir testweise das Fahrzeug seine engst mögliche Kurve fahren und legten deren doppelten Radius als Grenzwert für die Teststrecke fest. Diese Deklaration ist notwendig, damit der Roboter auch in der engsten Kurve zur Korrektur noch stärker einlenken kann.

## 4. Grundlagen K

Nachfolgend sind Hintergrundinformationen zu genutzten Vorgehensweisen in der Bildverarbeitung und der Navigation des TUCars zu finden.

### 4.1. Koordinatensysteme K

Alle Positionen von Objekten im Raum können mit Entfernungen relativ zu entsprechenden KS beschrieben werden. Für die Beschreibung unseres TUCars auf dem Parcours und die Lage der Straßenlinien wurden vier kartesische, hierarchisch angeordnete KS festgelegt, welche in Abbildung 4.1 verdeutlicht werden. Auch wenn sich der Roboter natürlich im dreidimensionalen aufhält, reicht eine 2D-Beschreibung für unsere Anwendung aus. Die Testumgebung beinhaltet keinerlei Höhenunterschiede. So lassen sich alle zu beschreibenden Punkte im Raum auf die Straßenebene projizieren.

Das Weltkoordinatensystem  $\mathcal{W}$  bildet die Basis und ist an einem beliebigen Punkt des Szenarios verankert. Es dient zur Beschreibung der Pose des mobilen Roboters und der Linienpunkte in der Weltkarte.

Die Pose bezeichnet den Zustandsvektor aus Position und Orientierung des Fahrzeugs in der Welt (siehe Gleichung (9.1)). Deren Lage bezeichnet das Roboterkoordinatensystem  $\mathcal{B}$ , welches in der Hinterachse des Autos festgelegt ist.

Damit sich die Pixelkoordinaten des Bildes nicht von der Matrixindizierung unterscheiden, wird das Bildkoordinatensystem  $\mathcal{I}$  in die linke obere Ecke des entzerrten Fotos gelegt. So zeigt die  $x^{\mathcal{I}}$ -Achse in Richtung der Zeilen und die Spalten laufen mit der  $y^{\mathcal{I}}$ -Koordinate.

Das Bild wird von der über dem Auto angebrachten Kamera aufgenommen. Die auf die Straßenebene projizierte Lage der Kamera wird als Kamerakoordinatensystem  $\mathcal{C}$  festgelegt, welches die Verbindung zwischen Roboter- und Bildkoordinatensystem darstellt.

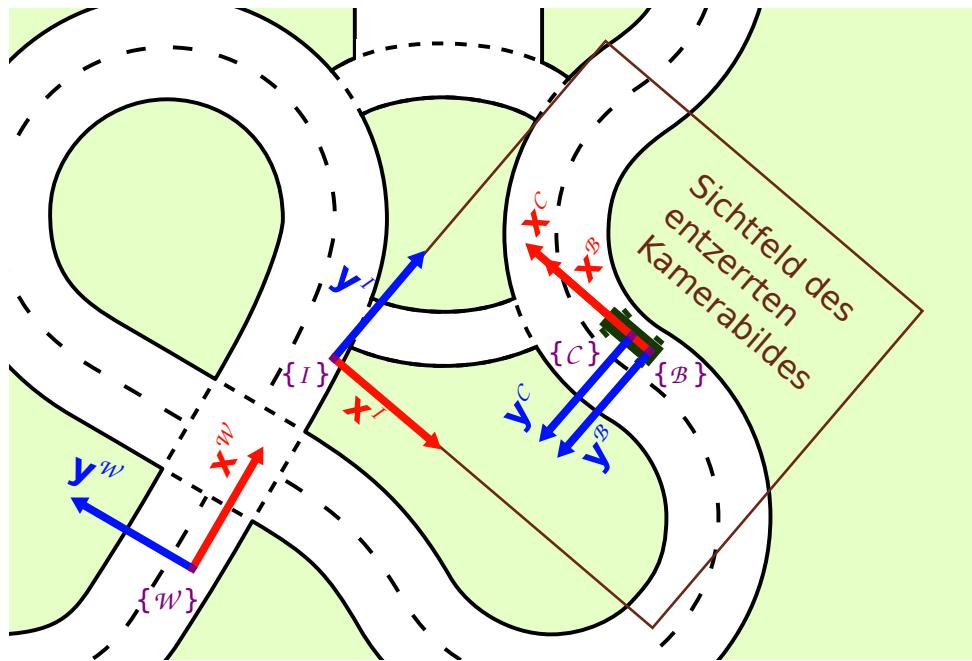


Abbildung 4.1.: Alle eingeführten Koordinatensysteme im Überblick

Für die festgelegten KS gelten zwei zum Einsatz gekommene Einheiten. Punkte im Welt- und Roboterkoordinatensystem werden in Millimetern und im Kamera- und Bildkoordinatensystem in Pixeln beschrieben. Ein Pixel entspricht ungefähr 3,9 Millimetern.

## 4.2. Koordinatentransformationen

Die eben eingeführten Koordinatensysteme dienen der Darstellung jeweils dazu passender Objekte. Oft ist dagegen auch deren Beschreibung in anderen KS gefragt. Um beispielsweise in Roboterkoordinaten erkannten Hindernissen deren Position in der Weltkarte zuschreiben zu können, bedarf es einer Umrechnung.

Für die Transformation von Punkten in ein anderes KS wird eine Verschiebung (Translation) in Richtung eines Translationsvektors  $t$ , eine Drehung mittels einer Rotationsmatrix  $R$  (Rotation) und gegebenenfalls eine Skalierung durchgeführt. Mathematisch lassen sich  $t$ ,  $R$  und die Skalierungsmatrix  $S$  wie folgt darstellen (Corke 2017, S. 26f), (Nischwitz 2011, S. 133).

$$\mathbf{t} = \begin{pmatrix} t_x \\ t_y \end{pmatrix}, \quad \mathbf{R} = \begin{pmatrix} \cos \delta & -\sin \delta \\ \sin \delta & \cos \delta \end{pmatrix}, \quad \mathbf{S} = \begin{pmatrix} \lambda_x & 0 \\ 0 & \lambda_y \end{pmatrix} \quad (4.1)$$

Da sich unser Roboter in der Darstellung nur im zweidimensionalen Raum bewegt, können Punkte lediglich in dieser Ebene rotiert werden (sprich: um die nicht vorhandene  $z$ -Achse gedreht werden). Sind beispielsweise die Weltkoordinaten von Punkten im Roboterkoordinatensystem gesucht, so stellen  $\mathbf{t}$  die Position und  $\delta = \theta$  die Orientierung des Modellfahrzeugs dar. Eine Abbildung des Punktvektors  $\mathbf{p}^{\mathcal{B}}$  in das  $\mathcal{W}$ -KS könnte dann wie folgt aussehen:

$$\mathbf{p}^{\mathcal{W}} = \mathbf{S}^{\mathcal{WB}} \cdot \mathbf{R}^{\mathcal{WB}} \cdot \mathbf{p}^{\mathcal{B}} + \mathbf{t}_{\mathcal{B}}^{\mathcal{W}} \quad (4.2)$$

Diese Kombination aus Rotation und Translation lässt sich ebenso in einer Matrix  $\mathbf{T}$  zusammenfassen, sodass man anstatt Gleichung (4.2) auch folgendes schreiben kann.

$$\mathbf{p}^{\mathcal{W}} = \mathbf{T}^{\mathcal{WB}} \cdot \mathbf{p}^{\mathcal{B}} \quad (4.3)$$

Die Matrix  $\mathbf{T}$  heißt homogene Transformationsmatrix und wird folgendermaßen aufgestellt (Corke 2017, S. 26f):

$$\mathbf{T} = \begin{pmatrix} \mathbf{S} \mathbf{R} & \mathbf{t} \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \lambda_x \cdot \cos \delta & -\lambda_x \cdot \sin \delta & t_x \\ \lambda_y \cdot \sin \delta & \lambda_y \cdot \cos \delta & t_y \\ 0 & 0 & 1 \end{pmatrix} \quad (4.4)$$

So lassen sich jegliche Transformationen zwischen aufeinanderfolgenden KS realisieren, wenn die Parameter Rotationswinkel  $\delta$ , Translationsvektor  $\mathbf{t}$  und die Skalierungsfaktoren bekannt sind. Für uniforme Skalierungen gilt  $\lambda_x = \lambda_y$ . Dabei ist eine Hierarchie<sup>1</sup> der KS sinnvoll, da sich z.B. die unbekannte Transformationsmatrix  $\mathbf{T}^{\mathcal{WI}}$ , welche Bildkoordinaten in Weltkoordinaten transformiert, durch Multiplikation bereits bekannter Matrizen ermitteln lässt. Es gilt:

$$\mathbf{T}^{\mathcal{WI}} = \mathbf{T}^{\mathcal{WB}} \cdot \mathbf{T}^{\mathcal{BC}} \cdot \mathbf{T}^{\mathcal{CI}} \quad (4.5)$$

Für die Transformation in umgekehrter Richtung kann die Inverse der homogenen

---

<sup>1</sup>Die Koordinatensystemhierarchie besitzt folgende Reihenfolge:  $\mathcal{W} \rightarrow \mathcal{B} \rightarrow \mathcal{C} \rightarrow \mathcal{I}$

Transformationsmatrix angewendet werden ( $\mathbf{T}^{\mathcal{IW}} = (\mathbf{T}^{\mathcal{WI}})^{-1}$ ). Ein Beispiel für die Transformation des Punktes  $\mathbf{p}^{\mathcal{B}}$  in Weltkoordinaten sieht folglich so aus:

$$\mathbf{p}^{\mathcal{W}} = \mathbf{T}^{\mathcal{WB}} \cdot \mathbf{p}^{\mathcal{B}} = \begin{pmatrix} \cos 30^\circ & -\sin 30^\circ & 420 \\ \sin 30^\circ & \cos 30^\circ & 170 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 230 \\ 350 \\ 1 \end{pmatrix} = \begin{pmatrix} 444, 1858 \\ 588, 1089 \\ 1 \end{pmatrix} \quad (4.6)$$

### 4.3. Filter in der Bildverarbeitung K

Das Ziel in der Bildverarbeitung ist meist die Extraktion von Informationen bzw. die Erkennung von Objekten. In unserem Fall gilt es hauptsächlich herauszufinden, an welcher Stelle sich Linien im Bild befinden, die zur Fahrspur gehören. Hierbei kann das Graustufenbild als Matrix verstanden werden, deren Einträge die Helligkeit der einzelnen Pixel darstellen.

Um bestimmte Eigenschaften zu ändern oder Informationen in einem Bild hervorzuheben, werden mathematische Operationen auf die Pixelmatrix angewendet. Diese Verfahren, deren Ergebnisse wieder ein Bild sind, teilen sich in Punktoperationen, Nachbarschaftsoperationen und globale Operationen auf (Jähne 2005, S. 111). Da Punktoperationen für jedes Pixel den Farb- oder Helligkeitswert neu berechnen, verwendet man sie typischerweise zur Korrektur von Kontrast, Helligkeit oder Farbraum.

Objekte, wie zum Beispiel eine Straßenmarkierung, zeichnen sich meist dadurch aus, dass sich Pixel in Merkmalen von ihren Nachbarpixeln unterscheiden. Bei den Nachbarschaftsoperationen berechnen sich die Werte der Pixel des neuen Bildes aus den Bildpunkten einer kleinen Umgebung um die Punkte. Weil durch eine Nutzung des Nachbarschaftsoperators das Bild verändert wird und Informationen generell verloren gehen, spricht man auch von einem *Filter*.

**Faltungsfilter** Das von uns verwendete und sehr verbreitete Faltungsfilter verrechnet zur Ergebnisbildung die Helligkeitswerte über einen Filterkern miteinander. Der Filterkern ist hierbei eine quadratische, symmetrische Matrix, die mit der Pixelmatrix des Bildes gefaltet wird. Die Faltung zweier Funktionen beschreibt den durch eine andere Funktion gewichteten Mittelwert einer Funktion. Unter dem Faltungsprodukt  $f_1(t) * f_2(t)$  zweier Originalfunktionen  $f_1(t)$  und  $f_2(t)$  versteht man allgemein

das Integral (Papula 2012, S. 584)

$$f_1(t) * f_2(t) = \int_{-\infty}^{+\infty} f_1(\tau) \cdot f_2(t - \tau) d\tau \quad (4.7)$$

Die Bildmatrix und der Filterkern sind jedoch diskrete Funktionen. Deswegen wird das Integral zu einer Summe. Seien  $M^*$  das gefilterte,  $M$  das originale Bild,  $F$  der Filterkern,  $k$  und  $l$  dessen Matrixdimensionen und  $(o_x, o_y)$  der Mittelpunkt des Filterkerns, dann ergibt sich folgende Berechnungsformel für die diskrete Faltung (Wikipedia 2018a):

$$M^*(x, y) = \sum_{i=1}^k \sum_{j=1}^l M(x - i + o_x, y - j + o_y) \cdot F(i, j) \quad (4.8)$$

Ein Faltungsfilter berechnet also den Wert des neuen Pixels aus dem gewichteten Mittelwert der umliegenden Pixel innerhalb des Filterkerns. Je nachdem, nach welcher Funktion gewichtet wird, ergibt sich der Name des Filters und was er in dem Bild verändert.

**Separierbarkeit** Um Rechenzeit mit einem zweidimensionalen Filter einzusparen, existiert die Möglichkeit, ihn so in zwei eindimensionale Operatoren zu zerlegen, dass deren Matrixmultiplikation wieder das ursprüngliche 2D-Filter ergibt ((4.9), aus (Jähne 2005, S. 123)).

$$\begin{pmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 4 & 6 & 4 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 4 \\ 6 \\ 4 \\ 1 \end{pmatrix} \quad (4.9)$$

Wenn dies möglich ist, nennt man das Filter *separierbar*. Die Anwendung des zweiten Operators auf das Zwischenergebnis des ersten auf das Bild hat das gleiche Ergebnis zur Folge, führt jedoch zu einem geringeren Rechenaufwand.

### 4.3.1. Gauß-Filter

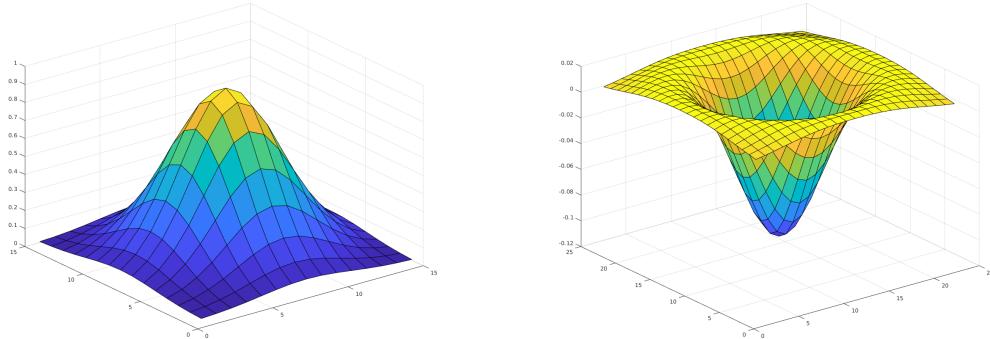
Der Filterkern des Gauß-Filters stellt sich aus der Dichtefunktion  $h$  der Normalverteilung auf („Gaußsche Glockenkurve“, (Papula 2016, S. 371)).

$$h(x) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{x^2}{2\sigma^2}} \quad (4.10)$$

Dieses Filter ist ein Tiefpassfilter und wird zur Glättung und zum Weichzeichnen verwendet. Hierbei wird angenommen, dass das Bildrauschen normalverteilt ist. Jenes kann mit diesem Filter verminder werden. Die zu diskretisierende Filterfunktion  $h_2$  in zwei Dimensionen ergibt sich, wenn man die Dichtefunktionen beider Richtungen  $x$  und  $y$  miteinander multipliziert.

$$h_2(x, y) = h(x) \cdot h(y) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (4.11)$$

Mit dem Zusammenspiel von der Filterkerngröße und der Standartabweichung  $\sigma$  lässt sich einstellen, wie stark das Bild verwischt. Je größer der Kern und  $\sigma$  sind, desto verschwommener ist das Resultat. In Abbildung 4.2a ist beispielhaft ein  $15 \times 15$  - Filterkern mit  $\sigma = 3$  dargestellt.



(a) 3D-Plot der Matrix eines  $15 \times 15$  - Gauß-Filters      (b) 3D-Plot der Matrix eines  $23 \times 23$  - LoG-Filters

Abbildung 4.2.: Dreidimensionale Visualisierung von Faltungsfilterkernen

### 4.3.2. Laplace-Filter

Möchte man in einem Bild Kanten detektieren, basiert die Lösung des Problems meist auf Ableitungen verschiedenen Grades (Jähne 2005, S. 360ff). (In der ersten Ableitung sind Kanten z.B. die Extremstellen.) Ein Filter zur Kantendetektion sollte Diskontinuitäten in den Grauwerten hervorheben und konstante Grauwerte unterdrücken. Der Laplace-Filter, bzw. diskrete Laplace-Operator, nutzt dafür die Summe der partiellen zweiten Ableitungen in alle Richtungen. (Papula 2016, S. 90)

$$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \quad (4.12)$$

Die Nulldurchgänge, vor und nach denen es unmittelbar Signalspitzen gibt, sind dann die gesuchten Kanten. Ein großer Nachteil dieser Filtermethode ist das relativ stark verrauschte Ergebnisbild.

Eine spezielle Form eines diskreten Laplace-Filters ist der „**Laplacian of Gaussian**“, oder auch Marr-Hildreth-Operator (LoG), welcher in dieser Arbeit zur Liniendetektion genutzt wurde. Der in Abbildung 4.2b beispielhaft dargestellte Filterkern wird durch die Anwendung des Laplace-Operators auf eine Gaußfunktion erstellt. Dieses Filter bewirkt neben der Kantendetektion eine gaußsche Glättung und mindert so das störende Rauschen.

## 4.4. Kalman-Filter M

Das Kalman-Filter ist ein bereits 1960 entwickeltes Verfahren zur Zustandsschätzung von zeitdiskreten, linearen Systemen (Kalman 1960). Durch seinen iterativen Aufbau, welcher alle vorherigen Zustände mit wenig Rechenaufwand berücksichtigen kann, ist es besonders interessant für Echtzeitanwendungen. Die Fähigkeit, redundante Messungen zu vereinigen, macht es in vielerlei Hinsicht sehr attraktiv. Die folgenden Grundlagen stellen eine Kurzfassung von (Marchthaler und Dingler 2017) dar.

### 4.4.1. Zustandsraumdarstellung

Um ein Kalman-Filter nutzen zu können wird für den betrachteten Prozess ein Modell im Zustandsraum benötigt. In (4.13) ist die allgemeine Zustandsraumbeschreibung für zeitinvariante, lineare Systeme aufgeführt. Da das Kalman-Filter für die

Verwendung mit zeitdiskreten Systemen konzipiert wurde und im digitalen Bereich Messwerte lediglich abgetastet vorliegen, muss dieses kontinuierliche Modell später diskretisiert werden (4.14).

$$\dot{\mathbf{x}}(t) = \mathbf{A} \cdot \mathbf{x}(t) + \mathbf{B} \cdot \mathbf{u}(t) \quad (4.13a)$$

$$\mathbf{y}(t) = \mathbf{C} \cdot \mathbf{x}(t) + \mathbf{D} \cdot \mathbf{u}(t) \quad (4.13b)$$

$$\mathbf{x}(k+1) = \mathbf{A}_d \cdot \mathbf{x}(k) + \mathbf{B}_d \cdot \mathbf{u}(k) \quad (4.14a)$$

$$\mathbf{y}(k) = \mathbf{C} \cdot \mathbf{x}(k) + \mathbf{D} \cdot \mathbf{u}(k) \quad (4.14b)$$

Auf den Indize  $d$  für die diskrete Systemmatrix  $\mathbf{A}$  sowie Eingangsmatrix  $\mathbf{B}$  wird im Folgenden verzichtet, da alle konsekutiven Systembeschreibungen diskreter Natur sind.

#### 4.4.2. Filtergleichungen

Sobald die Zustandsraumbeschreibung aufgestellt ist, können die folgenden Filtergleichungen angewandt werden:

##### 4.4.2.1. Korrektur

$$\hat{\mathbf{y}}(k) = \mathbf{C} \cdot \hat{\mathbf{x}}(k) + \mathbf{D} \cdot \mathbf{u}(k) \quad (4.15a)$$

$$\Delta\mathbf{y}(k) = \mathbf{y}(k) - \hat{\mathbf{y}}(k) \quad (4.15b)$$

$$\mathbf{K}(k) = \hat{\mathbf{P}}(k) \cdot \mathbf{C}^T \cdot (\mathbf{C} \cdot \hat{\mathbf{P}}(k) \cdot \mathbf{C}^T + \mathbf{R}(k))^{-1} \quad (4.15c)$$

$$\tilde{\mathbf{x}}(k) = \hat{\mathbf{x}}(k) + \mathbf{K}(k) \cdot \Delta\mathbf{y}(k) \quad (4.15d)$$

$$\tilde{\mathbf{P}}(k) = (\mathbf{I} - \mathbf{K}(k) \cdot \mathbf{C}) \cdot \tilde{\mathbf{P}}(k) \quad (4.15e)$$

##### 4.4.2.2. Prädikation

$$\hat{\mathbf{x}}(k+1) = \mathbf{A} \cdot \tilde{\mathbf{x}}(k) + \mathbf{B} \cdot \mathbf{u}(k) \quad (4.16a)$$

$$\hat{\mathbf{P}}(k+1) = \mathbf{A} \cdot \tilde{\mathbf{P}}(k) \cdot \mathbf{A}^T \quad (4.16b)$$

## 4.5. RANSAC M

In einem der in dieser Arbeit vorgestellten Ansätze zur Fahrspurverfolgung soll ein Modell, genauer ein Polynom 3. Grades, gefunden werden, welches den Verlauf einer Fahrbahnmarkierung möglichst exakt darstellt. Nachdem auf den Bilddaten eine Kantenextraktion ausgeführt wurde, stehen nun mögliche Kandidaten für Punkte der Fahrspurmarkierung fest. Durch ungünstige Beleuchtungsverhältnisse, verschmutzte Fahrbahnen oder andere Störeinflüsse enthalten diese Punktmenge oft Ausreißer, welche bei einer Regression durch die Methode der kleinsten Fehlerquadrate für falsche Modelle sorgen würden. „**RAN**dom **S**ample **C**onsensus“, deutsch etwa „Übereinstimmung mit einer zufälligen Stichprobe“ (RANSAC) (Fischler und Bolles 1981) ist ein Algorithmus um solche Ausreißer zu entfernen.

### 4.5.1. Ablauf des Algorithmus

In jeder Iteration wählt der Algorithmus zufällig mindestens so viele Punkte aus dem ihm übergebenen Datensatz, wie für die Bildung des Modells notwendig sind. Das Modell auf Basis dieser Punkte wird nun errechnet. Den nächsten Schritt stellt die Ermittlung des Abstandes der übrigen Koordinaten des Datensatzes vom gefundenen Modell dar. Punkte mit einem Abstand kleiner als ein festgelegter Schwellwert zählen als sogenannte Inlier. Ist das Verhältnis der Anzahl von Inliern zu Punkten mit Abstand größer als der definierte Grenzwert (Outlier) besser als in allen vorherigen Iterationen, wird das Modell vorgemerkt. Ist das Verhältnis von Inliern zu Outliern sogar besser als ein festgelegter Schwellwert, bricht der Algorithmus vor dem Erreichen einer maximalen Iterationsanzahl ab, andernfalls wird er bis zum Erreichen selbiger wiederholt. Abbildung 4.3 stellt die beschriebenen Schritte noch einmal anschaulich dar.

### 4.5.2. Nachfolgende Schritte

Das gefundene Modell mit den dazugehörigen, als Inlier gefundenen Punkten kann nun mit der Methode der kleinsten Quadrate weiter optimiert werden.

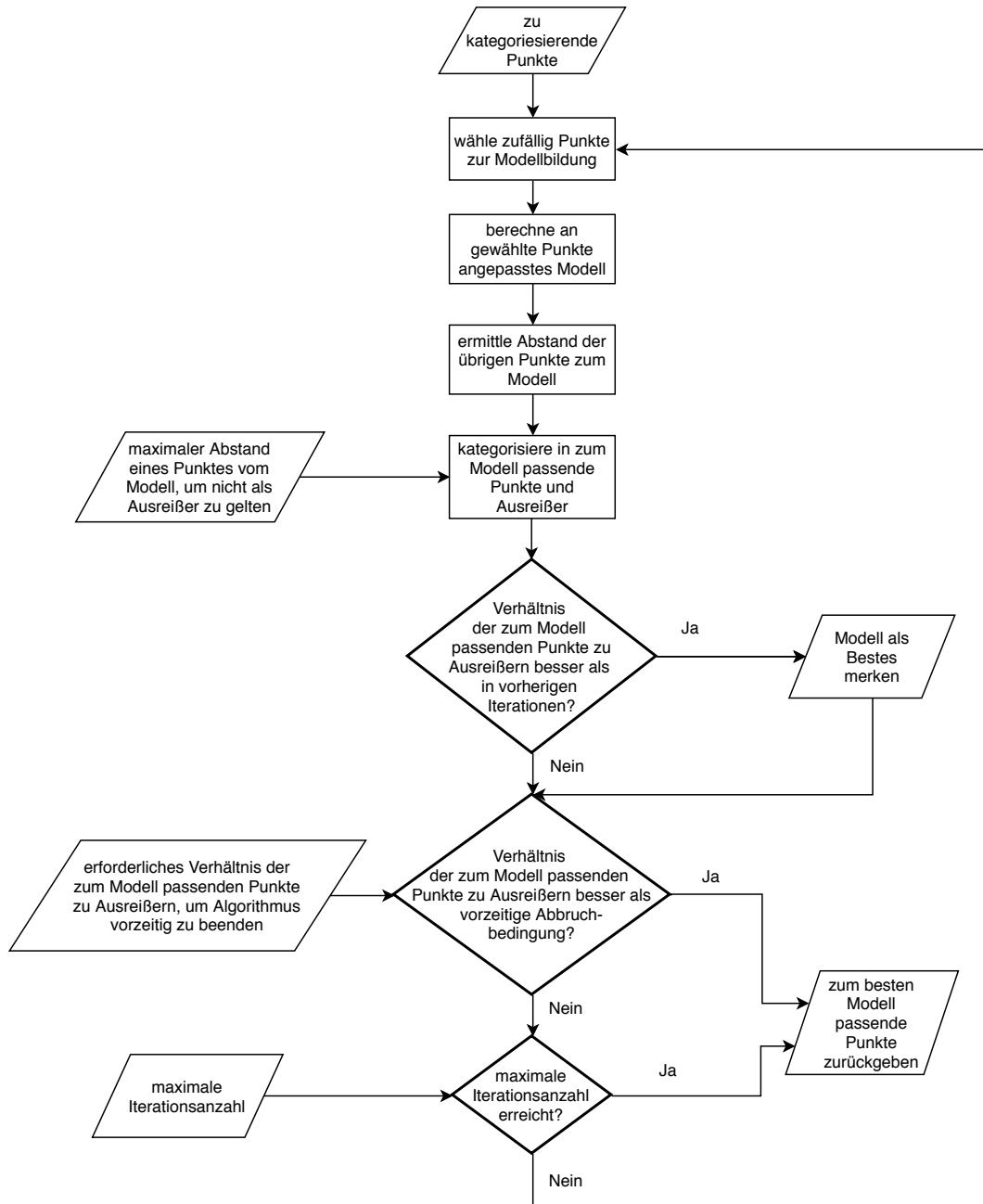


Abbildung 4.3.: Schema zum Ablauf des RANSAC-Algorithmus

## 4.6. Hough-Transformation K

Leicht parametrierbare geometrische Objekte in einem Binärbild (z.B. nach einer Kantenerkennung) lassen sich mit dem robusten Verfahren der Hough-Transformation finden. Dessen häufigste Anwendung besteht in der Geradenerkennung, weshalb das Prinzip im Folgenden an diesem einfachen Fall erklärt werden soll. Analog lässt sich die Vorgehensweise auf andere parametrische Figuren übertragen, wie zum Beispiel Kreise.

Um gerade Linien mittels Hough-Transformation in einem Bild finden zu können, ist ein Modell der Gerade, also die mathematische Beschreibung des zu findenden Objektes, notwendig. Alle Punkte  $(x,y)$ , die auf einer Geraden mit den Parametern  $m$  und  $n$  liegen, lassen sich nach folgender Gleichung beschreiben (Jähne 2005):

$$y = m \cdot x + n \quad (4.17)$$

Hierbei geben  $n$  den Achsenabschnitt und  $m$  den Anstieg der Geraden an. Die Geradengleichung (4.17) lässt sich nun so umstellen, dass der Anstieg  $m$  in Abhängigkeit vom Achsenabschnitt  $n$  dargestellt wird:

$$m = -\frac{1}{x} \cdot n + \frac{y}{x} \quad (4.18)$$

Damit wird ein zweiter sogenannter Modellraum aufgespannt, indem  $n$  und  $m$  zum Definitions- und Wertebereich gesetzt werden und die Punkte zu den Parametern zählen. Der Hough-Algorithmus transformiert jeden Bildpunkt in den Modellraum, welcher dort eine Gerade repräsentiert. Umgekehrt stellt ein Punkt im Modellraum eine Gerade im Datenraum dar. Viele im Bild auf einer gesuchten Geraden liegenden Pixel machen sich dann als häufiger Schnittpunkt im Modellraum bemerkbar. Jener Punkt  $(n,m)$  enthält eine solide Parameterabschätzung der gesuchten Gerade im Bild.

Praktischerweise wird als Modell für die Gerade die hessesche Normalform (Steigungswinkel und Abstand zum Koordinatenursprung als Parameter) eingesetzt, da die Beschreibung in Gleichung (4.17) den großen Nachteil hat, dass der Anstiegsparameter  $m$  bei vertikalen Geraden ins Unendliche läuft.

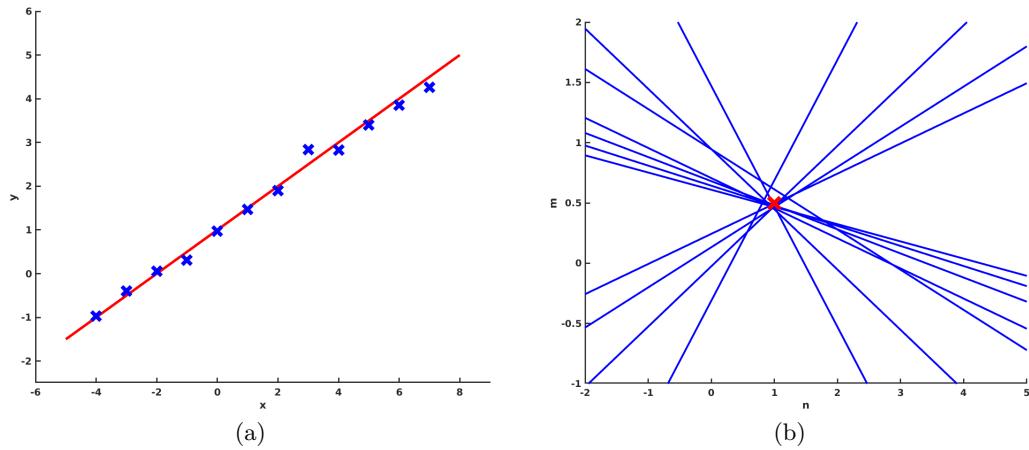


Abbildung 4.4.: Houghtransformation von Geraden: Punkte aus dem Datenraum (a) werden auf den Modellraum (b) abgebildet.

#### 4.6.1. Vereinfachte Hough-Transformation

Wir haben in unserer Arbeit in Anlehnung an (Aly 2008) eine sehr vereinfachte Form der Hough-Transformation zur Anwendung gebracht, welche lediglich zur Findung vertikaler Linien taugt. Dazu wird ein Histogramm über die Spaltensummen der Pixelmatrix gebildet. Nach der Glättung mittels Gauß-Filter geben die Lage der Maxima über einem bestimmten Schwellwert anschließend Auskunft über die Position einer näherungsweise vertikal erkannten Gerade.

### 4.7. Kameramodell M

Da das verwendete Fisheye-Objektiv durch seinen großen Blickwinkel stark verzerrt auf den Kamerasensor abbildet, ist ein spezielles Kameramodell zur Rektifizierung der Bilddaten notwendig. Wir haben uns in dieser Arbeit für die Nutzung einer Matlab-Toolbox<sup>2</sup> entschieden (Scaramuzza, Martinelli und Siegwart 2006a; Scaramuzza, Martinelli und Siegwart 2006b; Scaramuzza 2007; Rufli, Scaramuzza und Siegwart 2008). Das in dieser verwendete Modell und die Anwendung dessen zur Entzerrung soll nun erläutert werden.

Die Omnidirectional Camera Calibration (OCamCalib)-Toolbox behandelt das Ka-

<sup>2</sup><https://sites.google.com/site/scarabotix/ocamcalib-toolbox>

merasystem als Einheit, d.h. die Kamera und der Fisheyeobjektivaufschraub (alternativ der konvexe Spiegel) werden zusammen durch einen Parametersatz beschrieben.

Wie üblich, werden zu Beginn das Sensorkoordinatensystem  $\mathcal{S}$  im zweidimensionalen Raum und das Kamerakoordinatensystem  $\mathcal{C}$  in der dreidimensionalen Welt definiert. Das Modell hat das Ziel, den Zusammenhang zwischen einem Punkt auf dem Bildsensor  $s^{\mathcal{S}}$  mit Koordinaten  $(u, v)$  und einem vom optischen Zentrum des Objektivs ausgehenden Vektor nicht festgelegter Länge  $p^{\mathcal{C}}$  mit Koordinaten  $(p_x^{\mathcal{C}}, p_y^{\mathcal{C}}, p_z^{\mathcal{C}})$  zu finden (s. (4.21) und Abb. 4.5).

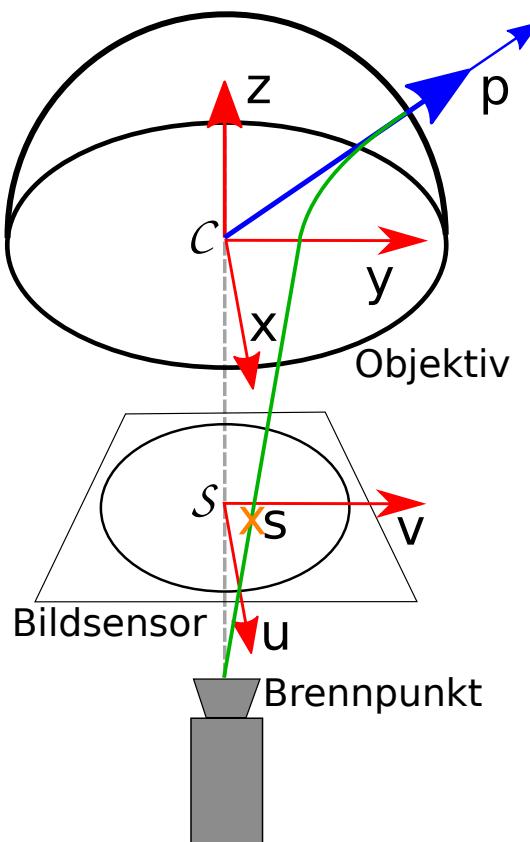


Abbildung 4.5.: Das Kameramodell der OCamlCalib-Toolbox

#### 4.7.1. Annahmen

Im verwendeten Modell werden folgende Annahmen getroffen:

1. Die verwendete Optik stellt ein zentrales Kamerasytem dar, d.h. alle einfal-lenden Strahlen, welche sich im Brennpunkt des Objektivs nach der Brechung schneiden (s. grüner Strahlverlauf in Abb. 4.5), schneiden sich auch ungebro-chnen in einem Punkt (s. blauer Strahlverlauf in Abb. 4.5). Dieser Punkt ist das optische Zentrum des Objektivs und der Koordinatenursprung des Kamerako-ordinatensystems  $\mathcal{C}$ .
2. Die Achse des Bildsensors und des Objektivs sind nahezu identisch (siehe un-terbrochene graue Linie in Abb. 4.5). Das Modell berücksichtigt nur kleine Abweichungen der Rotationen der Achsen gegeneinander.
3. Das Objektiv ist rotationssymmetrisch zu seiner Achse (s. unterbrochene graue Linie in Abb. 4.5).
4. Die Linsenverzerrung wird nicht durch herkömmliche Methoden, sondern durch die Projektionsfunktion  $f(u, v)$  (4.21) berücksichtigt.

#### 4.7.2. Herleitung

Da die Objektivachse und die Achse des Bildsensors in erster Näherung gleich sind, lässt sich schreiben:

$$\begin{pmatrix} p_x^{\mathcal{C}} \\ p_y^{\mathcal{C}} \end{pmatrix} = \lambda \cdot \begin{pmatrix} u \\ v \end{pmatrix} \quad \lambda \in \mathbb{R}^+ \quad (4.19)$$

Richtungs-Vektor  $\mathbf{p}^{\mathcal{C}}$  ergibt sich wie folgt:

$$\mathbf{p}^{\mathcal{C}} = \begin{pmatrix} p_x^{\mathcal{C}} \\ p_y^{\mathcal{C}} \\ p_z^{\mathcal{C}} \end{pmatrix} = \begin{pmatrix} \lambda \cdot u \\ \lambda \cdot v \\ f(u, v) \end{pmatrix} \quad (4.20)$$

Der Faktor  $\lambda$  kann in die Funktion  $f(u, v)$  integriert werden:

$$\mathbf{p}^{\mathcal{C}} = \begin{pmatrix} p_x^{\mathcal{C}} \\ p_y^{\mathcal{C}} \\ p_z^{\mathcal{C}} \end{pmatrix} = \begin{pmatrix} u \\ v \\ f(u, v) \end{pmatrix} \quad (4.21)$$

Da das Objektiv rotationssymmetrisch ist, kann  $f(u, v)$  nur vom Abstand eines Punktes zum Mittelpunkt des Bildsensors abhängig gemacht werden:

$$\rho = \sqrt{(u)^2 + (v)^2} \quad (4.22)$$

Aus (4.21) und (4.22) ergibt sich:

$$\mathbf{p}^C = \begin{pmatrix} p_x^C \\ p_y^C \\ p_z^C \end{pmatrix} = \begin{pmatrix} u \\ v \\ f(\rho) \end{pmatrix} \quad (4.23)$$

Die Aufgabe der Kalibration besteht nun im Ermitteln der Funktion  $f(\rho)$  (4.24), welche in der Toolbox durch ein Polynom 4. Grades approximiert wird:

$$f(\rho) = a_0 + a_1\rho + a_2\rho^2 + a_3\rho^3 + a_4\rho^4 \quad (4.24)$$

Da ein Darlegen des Prozesses zur Ermittlung der Parameter  $a_0 \dots a_4$  den Rahmen dieser Arbeit sprengen würde, wird darauf verzichtet.

### 4.7.3. Entzerrung

Mithilfe der gewonnenen Informationen kann die Entzerrung des Bildes nun stattfinden. Zuerst wird eine Ebene parallel zur  $p_x^C p_y^C$ -Ebene des Kamerakoordinatensystems  $\mathcal{C}$  erstellt. Die  $p_z^C$ -Koordinate ist hierbei wählbar. Diese Ebene wird nun in eine Menge von Punkten diskretisiert. Im entstandenen zweidimensionalen Feld kann jedem Element ein Vektor  $(p_x^C, p_y^C, p_z^C)$  zugeordnet werden (z.B. Punkt  $i^C$  in Abb. 4.6). Durch Gleichung (4.23) lassen sich diesem Vektor Koordinaten  $(u, v)$  auf dem KameraSensor zuordnen (s. Abb. 4.6). Das Pixel mit dem geringsten Abstand zu den gewünschten Koordinaten  $(u, v)$  liefert nun den Farbwert für jeden Punkt der Ebene, welche das entzerrte Bild darstellt.

### 4.7.4. Affine Transformation

Da Annahme 2 in der Realität nie exakt zutrifft und bei der Digitalisierung des Bildes durch den KameraSensor mit nichtquadratischen Pixeln zu rechnen ist, muss vor der Nutzung des hergeleiteten Modells eine affine Transformation der realen Koordinaten

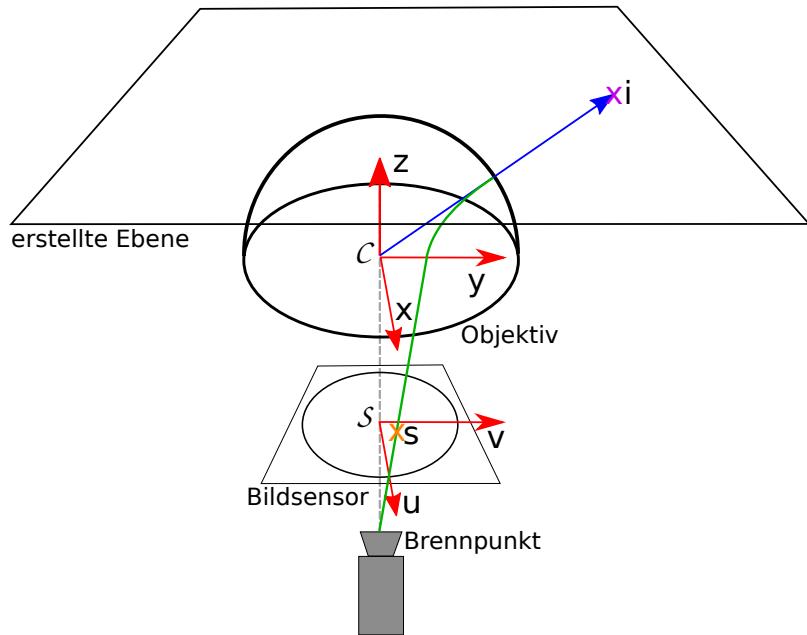
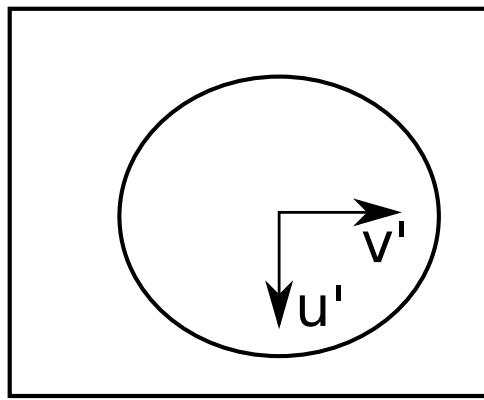


Abbildung 4.6.: Entzerrung mit dem Kameramodell der OCamCalib-Toolbox

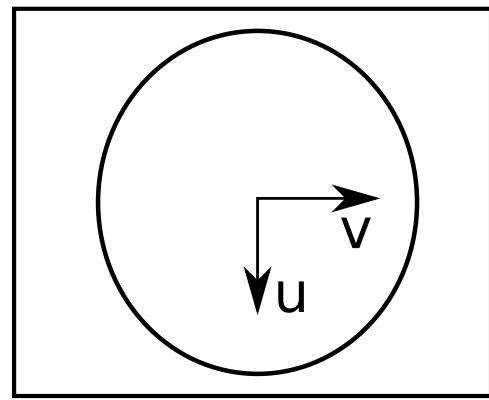
$(u', v')$  in die idealen Koordinaten  $(u, v)$  stattfinden (siehe Abb. 4.7).

$$\begin{pmatrix} u' \\ v' \end{pmatrix} = \begin{pmatrix} c & d \\ e & 1 \end{pmatrix} \cdot \begin{pmatrix} u \\ v \end{pmatrix} + t_{\mathcal{S}'}^{\mathcal{S}} \quad (4.25)$$

Die Verschiebung des Bildmittelpunktes (Koordinatenursprungs) von  $\mathcal{S}$  zu  $\mathcal{S}'$   $t_{\mathcal{S}'}^{\mathcal{S}}$ , sowie die Parameter  $c, d, e$  werden im Kalibrationsprozess durch die Toolbox ermittelt.



(a)



(b)

Abbildung 4.7.: Reales und Ideales Bildsensorkoordinatensystem  $\mathcal{S}'$  bzw.  $\mathcal{S}$

# 5. Struktur der Software

## 5.1. ROS

Da der Austausch von Bilddaten und Ansteuerbefehlen zwischen Modellfahrzeug und PC via ROS<sup>1</sup> abläuft, soll kurz darauf eingegangen werden.

ROS ist eine Sammlung von Werkzeugen, Bibliotheken und Konventionen zur Vereinfachung der Entwicklung komplexen und robusten Roboterverhaltens für eine Vielzahl an Roboterplattformen.<sup>2</sup>

### 5.1.1. Nodes

Ein ROS-System besteht aus mehreren Programmen „Nodes“, welche über bestimmte Kommunikationskanäle „Topics“ Datenpakete „Messages“ austauschen. Diese Applikationen können Information auf einem Topic zur Verfügung stellen, sie fungieren als sogenannte Publisher, oder nutzen, in diesem Fall man nennt sie Subscriber. Ein Schema zum Aufbau des konkreten ROS-Systems findet sich in Abb. 5.1.

#### 5.1.1.1. Fahrzeug

**Kamera-Node** Der Kamera-Node veröffentlicht die Bilder als Rohdaten-Topic sowie in komprimierter Form.

**Fahrzeug-Node** Der Fahrzeug-Node veröffentlicht:

- Odometriedaten
- Messwerte der IMU

---

<sup>1</sup><http://www.ros.org/>

<sup>2</sup>Übernommen aus dem Englischen von <http://www.ros.org/about-ros/> am 11. September 2018

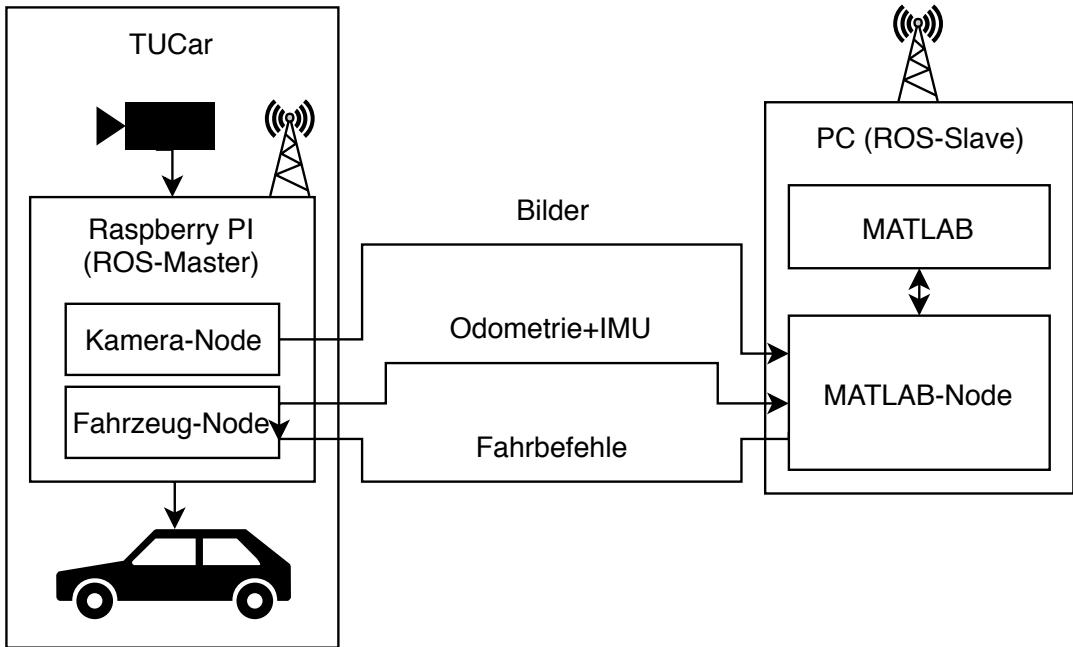


Abbildung 5.1.: Struktur des ROS-Systems

- sonstige Informationen zum Fahrzeug (z.B. Akkuspannung)

und nimmt im Ansteuer-Topic die gewünschte Geschwindigkeit und den Lenkwinkel entgegen.

### 5.1.1.2. PC

Auf dem PC wird nur der von MATLAB automatisch initialisierte Node genutzt. Dieser abonniert das komprimierte Bilddaten-Topic und veröffentlicht auf das Ansteuer-Topic des Fahrzeugs.

## 5.2. MATLAB

Die im Hauptteil dieser Arbeit beschriebenen Algorithmen zur Fahrspurverfolgung wurden ausschließlich in MATLAB implementiert. **MAT**rix **L**ABoratory ist ein kommerzielles Programm und eine Programmiersprache für numerische Rechnungen und grafische Darstellungen, dessen Grundbausteine Matrizen sind. Diese Eigenschaften, weitreichende Debugging-Funktionalitäten und die einfache Erweiterbarkeit

durch sogenannte Toolboxen machen es sehr attraktiv zum Einsatz im Rahmen der Bildverarbeitung & Evaluation. Da die grundlegende Softwarestruktur in allen drei Lösungsansätzen zur Fahrspurverfolgung gleich ist, soll diese nun dargelegt werden. Eine schematische Darstellung ist in Abb. 5.2 gegeben.

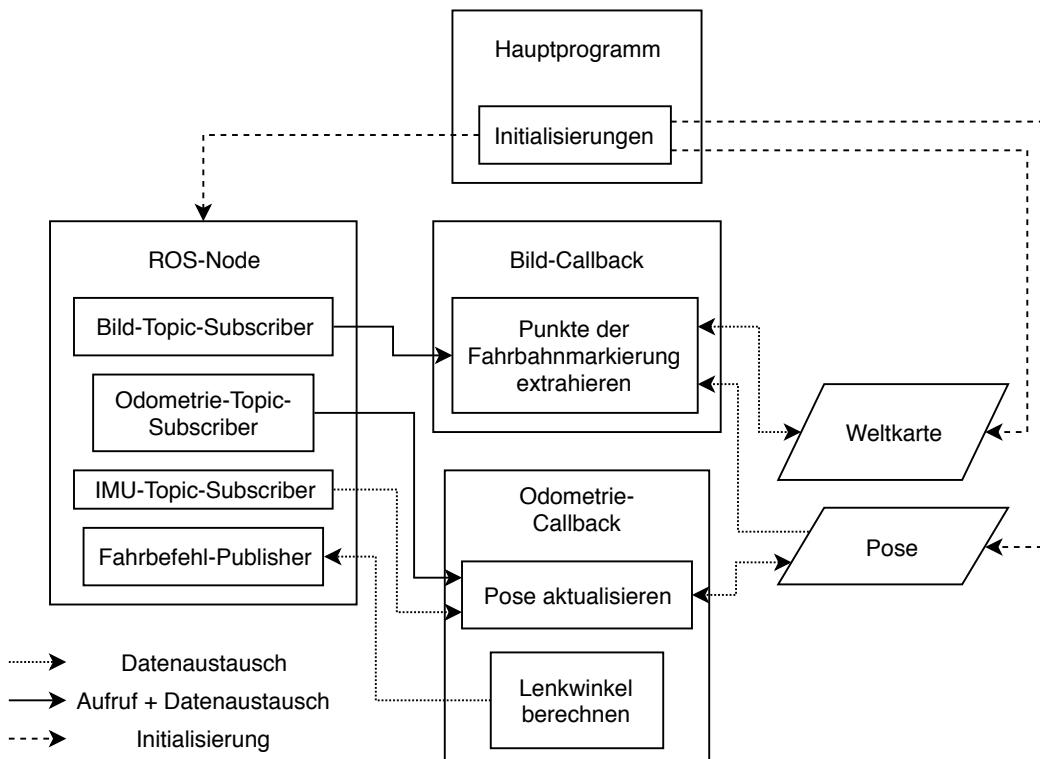


Abbildung 5.2.: Struktur des MATLAB-Softwareteils

### 5.2.1. Initialisierung

Ist das Fahrzeug betriebsbereit, d.h. werden alle Nodes des Fahrzeugs ausgeführt, so kann in MATLAB das Programm zur Fahrspurverfolgung aufgerufen werden. Hierfür wird die Initialisierungsroutine gestartet welche:

- die globalen Parameter lädt
- die Weltkarte und die Pose initialisiert
- den MATLAB-ROS-Node startet

- die Subscriber des Bilddaten-, Odometrie- und IMU-Topics anlegt
- eine konstante tangentiale Geschwindigkeit des Fahrzeugs vorgibt

### 5.2.2. Callbacks

**Odometrie-Callback** Das Fahrzeug veröffentlicht in konfigurierbaren Abständen seine aktuellen Odometriedaten. Das Eintreffen eines solchen Datenpaketes in MATLAB führt zum Aufrufen der Odometrie-Callback-Funktion. Die Hauptaufgabe dieses Programmteils besteht in der Regelung des Fahrzeugs, deren Ablauf im folgenden kurz erläutert wird.

1. Mithilfe der empfangenen, seit der letzten Odometrie-Message verstrichenen Zeit, der in dieser Spanne ermittelten Durchschnittsgeschwindigkeit und der durch die IMU erhaltenen Orientierung kann die Pose aktualisiert werden. Vorherige Posen werden inklusive des Zeitstempels der sie generierenden Odometrie-Daten für den im Folgenden erläuterten Bild-Callback abgespeichert.
2. Auf Basis der neuen Pose kann mithilfe aller in der Weltkarte vorhandenen Punkte ein neuer Lenkwinkel bestimmt werden.

**Bild-Callback** Sobald ein neues Bild im zugehörigen Topic verfügbar ist, wird ein Algorithmus zur Extraktion von Punkten der Fahrbahnmarkierung gerufen. Dieser fügt die gefundenen Koordinatenserien anhand der am besten zum Aufnahmepunkt des Bildes passenden Pose nach Linientyp sortiert („linke“, „mittlere“ oder „rechte“ Linie) in die Weltkarte ein und macht sie so für die Regelung abrufbar.

Eine ausführlichere Erläuterung zur Wirkungsweise und zu Vorteilen einer Aufteilung von Bildverarbeitung und Regelung in mehrere Callback-Funktionen ist in Abschnitt 9.4 gegeben.

### 5.2.3. Multitasking

Da MATLAB ohne spezielle Toolboxen Prozesse nicht gleichzeitig ausführen kann, die Bearbeitungszeit eines Bild-Callbacks jedoch wesentlich länger ist als die angestrebte Periodendauer der Regelung (Häufigkeit des Aufrufs eines Odometrie-Callbacks), musste für eine Unterbrechbarkeit der Bildverarbeitung gesorgt werden. MATLAB

reagiert bei der Ausführung eines Tasks nur während bestimmter Befehle auf weitere Ereignisse. Eine Möglichkeit die Regelung auch bei noch nicht vollständig ausgeführtem Bild-Callback zu ermöglichen, besteht im Einfügen kurzer Pausen zwischen Programmteilen zur Segmentierung dessen. Wählt man diese Abschnitte hinreichend kurz, kann der wartende Odometrie-Callback mit wenig Verzögerung ausgeführt und die Bildverarbeitung danach fortgesetzt werden.

## 6. Bildvorverarbeitung K

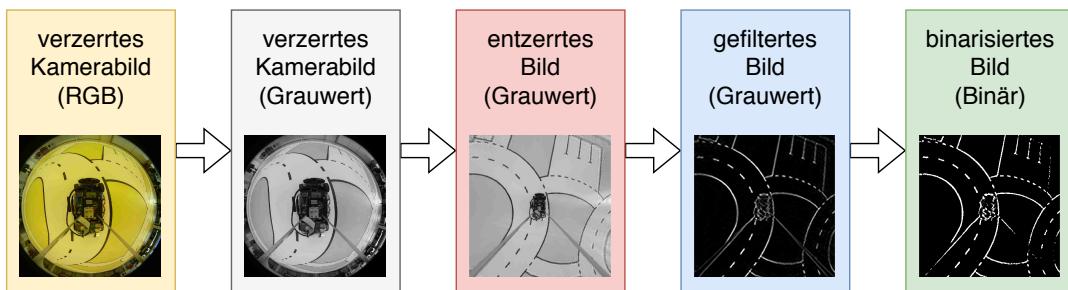


Abbildung 6.1.: Schritte in der Bildvorverarbeitung

Ein Bild ist eine Matrix aus Punkten, denen ein Farbwert zugeordnet wird. In der Bildverarbeitung gilt es in diesen farbigen Punkten enthaltene Informationen zu extrahieren. Der einzige uns zur Verfügung stehende Sensor ist neben den Radencodern und der IMU die in Vogelperspektive montierte Fischaugenkamera. Ziel nach der Aufnahme des Bildes ist es, Auskunft über die Position von Fahrspurmarkierungen zu erhalten, damit weitere Algorithmen (in Kapitel 7 beschrieben) daraus die für die Regelung relevanten Punkte detektieren können. Letztlich werden die Pixel gesucht, welche einer Linie zugehörig sind. Der menschliche Betrachter des Bildes erkennt die Linie sofort. Für die automatisierte Extraktion von Markierungspunkten bedarf es jedoch einer dem Rechner verständlichen Definition, wann ein Pixel einer Linie zugehörig ist. Dafür werden Linienpunkte durch die folgend beschriebenen Bildverarbeitungsschritte hervorgehoben, damit das Kriterium für den Rechner lautet: Alle Pixel, welche heller als ein bestimmter Schwellwert sind, gehören zu den Fahrbahnmarkierungspunkten. Zum Überblick sind die Verarbeitungsstufen vom Rohbild bis zum Endergebnis in Abbildung 6.1 dargestellt.

## 6.1. Bildentzerrung

Der große Vorteil des Fischaugenobjektivs ist es, Bereiche in größerer Entfernung aller Richtungen wahrnehmen zu können. Andererseits ist das Bild aufgrund des enormen Weitwinkels stark verzerrt. Zur Erstellung einer Weltkarte ist es daher sinnvoll, dessen Verzerrung zu rektifizieren. Dazu wurde sich in MATLAB wie in Kapitel 4.7 beschrieben der OCamCalib-Toolbox bedient. Da die Farbinformationen der Rohdaten keine Anwendung finden und sowohl die Toolbox als auch die Faltungsfilter lediglich auf Graustufenbildern arbeiten können, erfolgt vor der Entzerrung eine Umwandlung in ein Grauwertbild.

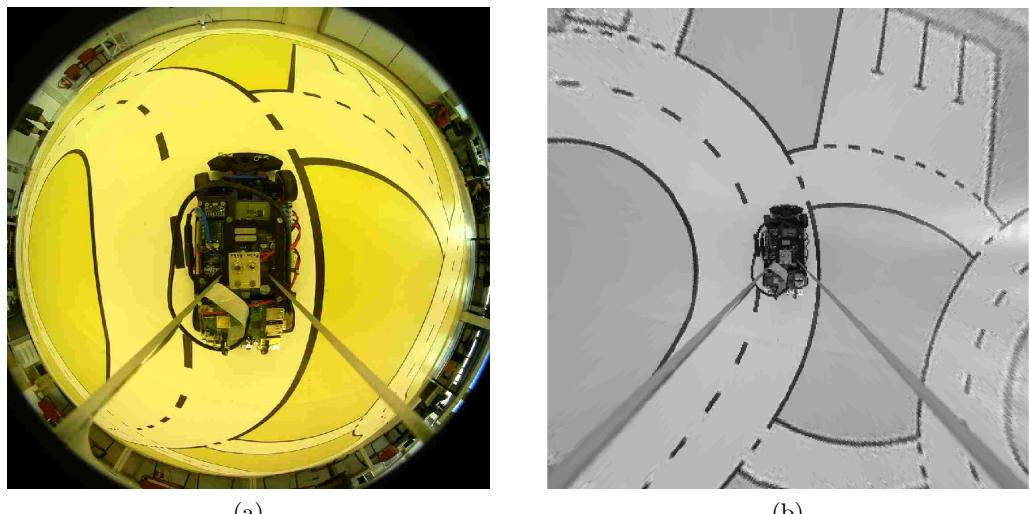


Abbildung 6.2.: verzerrtes Rohbild (a) und entzerrtes Graustufenbild (b) einer Momentaufnahme im Parcours

### 6.1.1. Determination der Auflösung

In Abbildung 6.2 ist ein Bild original (links) und nach der Entzerrung (rechts) zu sehen. Im entzerrten Bild fällt auf, dass zum Rand die Auflösung durch dessen Transformation zunehmend schlechter wird. Diese Tatsache lässt den Schluss zu, für die Kamera die vorzugsweise vollste Auflösung zu wählen, mit dem Ziel, die Informationsdichte am Bildrand so hoch wie möglich zu halten. Damit einher geht allerdings eine längere Übertragungszeit des Bildes vom Roboter zum Rechner. Um diese Zeit

durchführbar gering zu halten, gilt es die niedrigste Auflösung bei möglichst uneingeschränkter Funktionalität zu bestimmen.

Das dafür entscheidende Kriterium ist das kleinste, zu detektierende Objekt, welches in seiner Größe zwei Pixel nicht unterschreiten darf. In unserem Fall ist dies die Breite einer Fahrbahnmarkierung im entzerrten Bild. Aus Sicherheitsgründen ist die Auflösung nach der Entzerrung mit  $400 \times 400$  Pixeln so gewählt, dass die Spurmarkierung eine Dicke von ca. 4-6 Pixeln aufweist.

Durch die als ausreichend festgelegte Sichtweite ergibt sich die Auflösung des verzerrten Kamerabildes. Sie wurde empirisch auf  $1080 \times 1080$  so eingestellt, dass nach der Filterung des entzerrten Bildausschnitts alle Linien bis zum Rand akzeptabel erkennbar geblieben sind.

## 6.2. Filterung

Die Aufnahme kann nun wie in Kapitel 4.3 beschrieben gefiltert werden, damit sich die Kanten (sprich: steile Übergänge von hellen zu dunklen Pixeln oder andersherum) hervorheben. Monotone Bildbereiche sollen dunkel bleiben. Unter Durchführung einer Kantendetektion auf dem entzerrten Bild erhielt man sehr viele Treffer, da durch sensorbedingtes Bildrauschen den Kantendetektor ansprechende Störungen enthalten sind.

Das in Abbildung 6.3 (a) dargestellte Filterergebnis wurde mit einem LoG-Filter erzielt. Dieser vereint mit der gaußschen Glättung und dem Laplace-Operator die Ausführung zweier Filter. Wie in Abschnitt 4.3.2 erklärt, stellen Kanten in der Filterantwort die Nulldurchgänge zwischen den positiven und negativen Peaks dar. Da das gefilterte Bild keine Pixelwerte kleiner Null zugewiesen bekommt, sind nur deren stark positiven Anstiegsänderungen ( $2.$  Ableitung  $> 0$ ) in Abb. 6.3 (a) sichtbar.

Im Allgemeinen treten an einer Linie zwei Kanten auf. Jedoch lassen sich mit entsprechend eingestellten Parameterwerten die zwei an jeder Fahrbahnmarkierung entstandenen positiven Peaks zu einer sichtbaren, die Fahrbahnbegrenzung repräsentierenden, hellen Linie verwischen.

Um wertvolle Rechenzeit während der aufwändigen Filterung einzusparen, werden das Bild mit je einem eindimensionalen  $15 \times 1$  bzw.  $1 \times 15$  Filter in  $x^T$ - und  $y^T$ -Richtung verarbeitet und anschließend die Filterergebnisse addiert. Diese Alternative zu der in Abschnitt 4.3 beschriebenen Separierbarkeit hat ebenfalls einen deutlich schnelleren

Funktionsablauf ermöglicht. Erstaunlicherweise ist die Qualität des gefilterten Bildes sehr gut, sodass die Weiterverarbeitung verlässliche Informationen liefert.

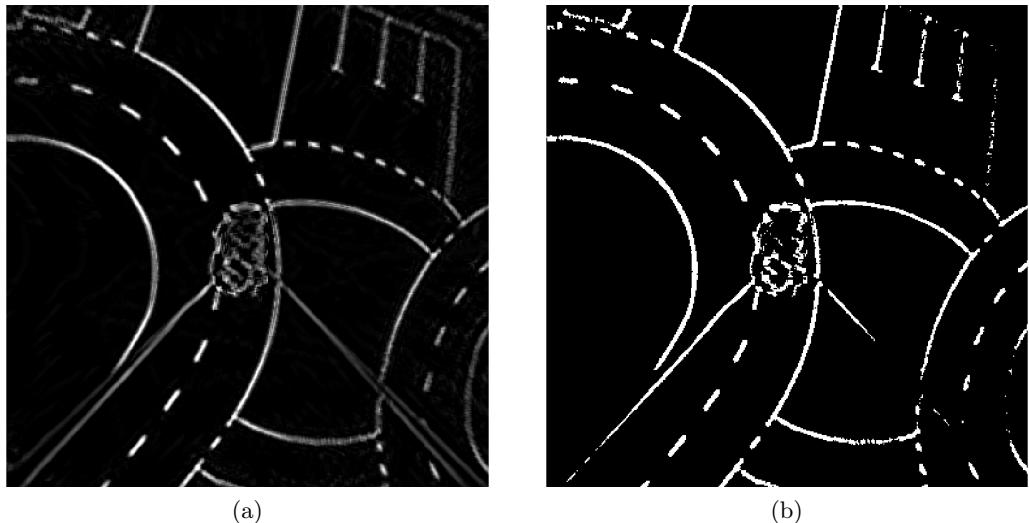


Abbildung 6.3.: gefiltertes (a) und anschließend binarisiertes Bild (b) einer Momentaufnahme im Parcours

### 6.3. Binarisierung

Der Schwellwert, welcher entscheidet, ob ein Pixel im binarisierten Bild den Wert „0“ oder „1“ erhält, wurde fest als Parameter eingestellt. Dessen Ermittlung ging aus der empirischen, manuellen Sichtprüfung in Abstimmung der Filterparameter hervor. Alle weiteren Bildverarbeitungsalgorithmen ziehen das binarisierte oder gefilterte Bild als Grundlage ihrer Berechnungen heran.

## 7. Polynomisierte Fahrspurerkennung

M

Ein zentrales Ziel dieser Arbeit stellt die Erkennung von Fahrspuren in einem Kamerabild dar. Mithilfe der im vorherigen Kapitel beschriebenen Bildvorverarbeitung wurde bereits eine Extraktion der für die folgenden Algorithmen notwendigen Eingangsinformationen vorgenommen. Die grundsätzliche Aufgabe der Konzepte zur Fahrbahndetektion besteht in einer weiteren Reduktion und Sortierung dieser vorverarbeiteten Daten zu einem mathematischen Modell der Straße. Die Komplexität dieser Darstellung variiert in der Literatur sehr stark, in (Narote u. a. 2018) wird ein Überblick zu möglichen Modellen gegeben. Einige simple Repräsentationen wurden von Beginn der Recherche an ausgeschlossen, da sie dem vorgegebenen Szenario nicht gerecht werden:

- Die deutliche Krümmung der Kurven unserer Teststrecke lässt eine Annäherung des Straßenverlaufs durch eine Gerade nicht zu. Ein komplexeres Modell (mehr als zwei Parameter) schließt die oft verwendete, fehlerresistente Hough-Transformation (s. Abschnitt 4.6) weitgehend aus, da ein mehr als zweidimensionaler Modellraum zu rechenaufwendig für eine Echtzeitanwendung ist.
- Da auch eine Änderung der Krümmung (S-Kurve) des zu approximierenden Fahrspurverlaufs abgebildet werden soll, kann auch das oft verwendete Polynom zweiten Grades den Anforderungen nicht genügen.

Die ersten zwei im Folgenden beschriebenen Verfahren bedienen sich deshalb eines Polynoms 3. Grades zur Modellierung der Fahrbahn bzw. deren Markierungen.

**Linienkoordinatensystem** K Zwischenzeitlich wurde zusätzlich zu den in Kapitel 4.1 beschriebenen KSs ein Linienkoordinatensystem  $\mathcal{L}$  festgelegt. Es plazierte

sich so in der unteren rechten Bildecke, dass dessen  $x^{\mathcal{L}}$ -Achse parallel zu der erwarteten Fahrbahnmarkierung verläuft, wenn das Auto auf einem geraden Abschnitt fährt. Der Grund für dieses weitere KS war die Beschreibung der approximierten Fahrspur mit einem Polynom dritten Grades (siehe Gleichung 7.1a) und eine zuerst anders angedachte Orientierung der angebrachten Kamera. Die Fahrspur darf im KS keine Parallele zur Ordinate darstellen, da sie in diesem Fall schlecht durch ein Polynom hätte approximiert werden können. In diesem Kapitel beziehen sich alle Funktionen zur approximierten Beschreibung der Fahrspur auf dieses Linienkoordinatensystem  $\mathcal{L}$ .

## 7.1. Ransac mit Maskierung K

Aus einem Foto mittels RANSAC eine mathematische Approximation einer Fahrbahnmarkierung zu erhalten, war die von uns anfangs angestrebte Herangehensweise. Der nun folgend beschriebene Ansatz zur Detektion der Fahrbahnlinien ist jedoch in der endgültigen Implementierung nicht zum Einsatz gekommen. Da wir uns jedoch geraume Zeit damit beschäftigt und viele später weiter verwendete Grundfunktionen aufgebaut haben, sollen die Vorgehensweise und die zum Ausschluss der Methode geführten Probleme näher erläutert werden.

### 7.1.1. Approximation der Straßenmarkierungen

Ausgangspunkt für die Ermittlung einer passenden Funktion ist das binarisierte Bild, welches nach den in Abbildung 6.1 dargestellten Schritten der Bildvorverarbeitung vorliegt. Alle Pixel mit dem Wert „1“ stellen im Idealfall potenzielle Linienpunkte dar. Diese sind der richtigen Kategorie „linke“, „mittlere“ oder „rechte“ Linie zuzuordnen, wenn sie nicht bereits mittels RANSAC-Algorithmus als Outlier (siehe Abschnitt 4.5.1) markiert wurden. Dazu wird je nach Zugehörigkeit eine entsprechende „Region of Interest“, deutsch: „Bereich von Interesse“ (RoI) im Bild festgelegt. Anschließend wird der in Kapitel 4.5.1 beschriebene RANSAC-Algorithmus in dem durch die RoI bestimmten Bildausschnitt durchgeführt.

Wie in Abbildung 7.1 zu sehen, folgt die Approximation dem Verlauf der Fahrbahnlinien relativ gut. Die eingezeichnete grüne Funktion beschreibt das Ergebnis des Fit eines Polynoms dritten Grades durch die von RANSAC bestimmten Inlier. Mit

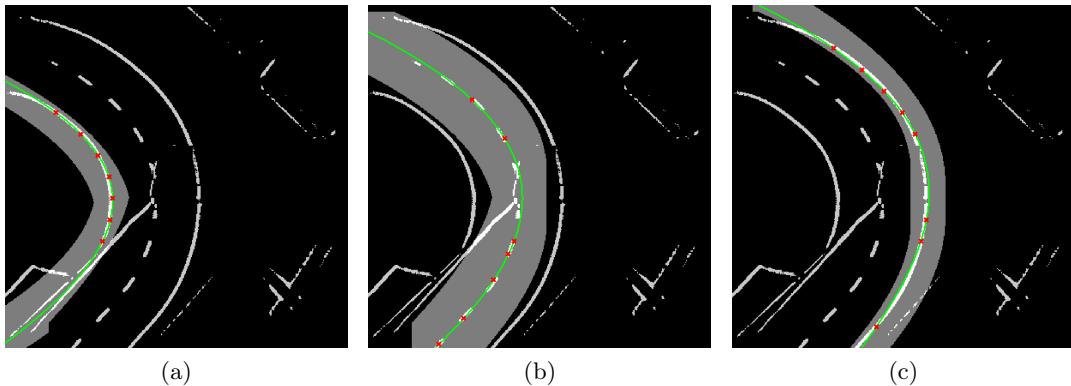


Abbildung 7.1.: Approximation der linken (a), mittleren (b) und rechten (c) Fahrbahnmarkierung durch ein Polynom dritten Grades mithilfe jeweiliger RoIs und der Anwendung von RANSAC

einem gewissen Abstand verteilt werden anschließend auf dem Polynom potentielle Kartenpunkte diskretisiert, welche einzig dann in die Karte aufgenommen werden, wenn sie sowohl Teil der Funktion sind als auch mit weißen Pixeln des maskierten Bildes übereinstimmen. So wird verhindert, Punkte an stark abweichenden Stellen der Approximation zur echten Fahrbahnmarkierung in die Weltkarte einzutragen.

### 7.1.2. Bestimmung der „Regions of Interest“

Um die zu einer Linie zugehörigen Pixel an der erwarteten Stelle suchen zu können, wird für jede der drei Fahrbahnmarkierungen eine passende Maske erstellt, welche den Bildausschnitt für RANSAC markiert. Diese RoIs werden mittels der bereits eingetragenen Kartenpunkte und der Pose zum Zeitpunkt der Bildaufnahme gebildet. Dazu werden eine bestimmte Anzahl der zuletzt eingetragenen Kartenpunkte gleichartiger Kategorie in das aktuelle Bild transformiert, welche folgend die Grundlage für eine Regression darstellen. Die daraus ermittelte Funktion bildet die Mitte der RoI. Nach einer Dilatation des im Bildausschnitt liegenden Teils der Funktion um einen Parameterwert entsteht ein dickes Band aus Punkten, welche die RoI beschreiben (siehe Abbildung 7.1).

### 7.1.3. Initialer Maskenbau

Ein nicht zu vernachlässigendes Problem stellt allerdings die Initialisierung dar. Wie gewinnt man die Masken, wenn bisher keine Linienpunkte in die Karte eingetragen worden sind? Die einfachste und am Anfang auch implementierte Möglichkeit bietet die aus Parameterwerten vorgeschriebene Maske einer Gerade, in der Annahme, das Auto starte immer auf einem geraden Abschnitt. Da so aber die Robustheit der Linienerkennung schlecht gewährleistet wäre, wurde dies schnell verworfen und eine bessere Möglichkeit gefunden.

**Rol für die Mittellinie** Die gestrichelte Mittellinie besitzt im gesamten, binarisierten Bild Alleinstellungsmerkmale, die sich durch ein darauf angepass tes, weiteres Faltungsfilter nutzen lassen. Die Idee für den in Abbildung 7.2 gezeigten Filterkern wurde aus (Drausche ke 2016) entnommen. Der im folgenden als „Ringfilter“ bezeichnete Kern wurde aus einem Innen- und Außenkreis entworfen, sodass der Radius des inneren Kreises mindestens so groß wie die Länge

eines Mittellinienstriches im Bild und der Außenkreisradius maximal so groß wie der kürzeste Abstand zwischen zwei Mittellinienstrichen ist. Was als Filterergebnis beispielhaft entsteht, ist in (c) der Abbildung 7.3 zu erkennen. Das Filter bewirkt, dass im Ergebnisbild genau dann ein Pixel mit „1“ beschrieben wird, wenn der Filterring um dieses Pixel mindestens einen Punkt im binarisierten Bild schneidet. Es entstehen abgegrenzte Bereiche mit einer „0“ als Filterantwort genau dann, wenn Mittellinienstriche innerhalb des „Ringfilters“ liegen und kein weiteres Pixel auf dem Ring um diese Striche gefunden wird.

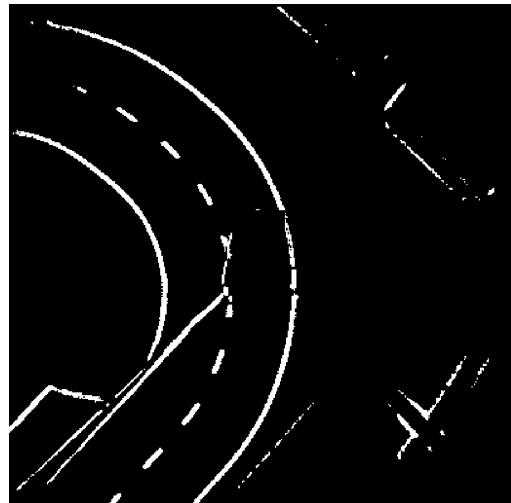
Die Negation des Filterergebnisses als Maske auf das binarisierte Bild gelegt erzeugt eine Grafik, die allein Pixel aus Punktgruppen enthält, die dem „Ringfilter“ entsprechend festgelegte Maximallängen und Mindestabstände zu anderen Punkten besitzen. Darauf kann nun der RANSAC-Algorithmus angewendet werden.



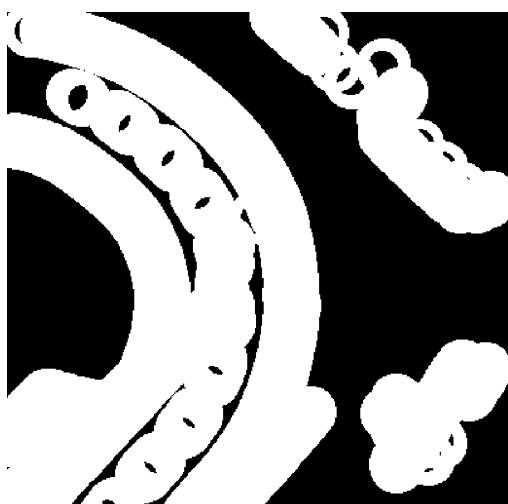
Abbildung 7.2.: Der Kern des „Ringfilters“



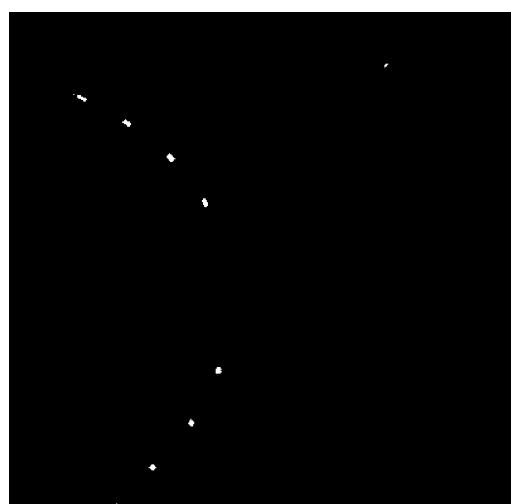
(a)



(b)



(c)



(d)

Abbildung 7.3.: entzerrtes (a) und binarisiertes Bild (b) einer Testaufnahme auf der Strecke in alten Kameraeinstellungen; Filterergebnis des „Ringfilters“ (c) und Durchschnitt dessen Negation mit dem binarisierten Bild (d)

**Randlinienmasken** Die zwei initialen RoIs für die Randlinien werden anschließend aus der approximierten Funktion dritten Grades der Mittellinie gebildet. Dies geschieht aus der gerichteten Verschiebung diskreter Punkte der Approximation. Gleichung (7.1a) sei eine mathematische Beschreibung der Funktion, die den Linienverlauf approximiert, dann ist Gleichung (7.1b) die Ableitung, welche den Anstieg an den Stellen  $x$  beschreibt. Der Verschiebungsvektor  $\mathbf{d}$  steht dann senkrecht auf  $f$ .

$$f(x) = ax^3 + bx^2 + cx + d \quad (7.1a)$$

$$f'(x) = 3ax^2 + 2bx + c \quad (7.1b)$$

$$\mathbf{d} = \frac{1}{\sqrt{(-f'(x))^2 + 1}} \cdot \begin{pmatrix} -f'(x) \\ 1 \end{pmatrix} \quad (7.1c)$$

$$\begin{pmatrix} x_s \\ y_s \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} \pm b_F \cdot \mathbf{d} \quad (7.1d)$$

Die Normierung von  $\mathbf{d}$  multipliziert mit der bekannten Fahrspurbreite  $b_F$  addiert auf den Mittellinienpunkt  $(x, y)$  ergibt einen Stützpunkt  $(x_s, y_s)$  der linken (+) bzw. der rechten (-) Randlinie (Gleichungen (7.1c) und (7.1d)).

Durch die so verschobenen Stützpunkte der erwarteten Lage der Randlinien werden ein neuer Fit und eine anschließende Dilatation vorgenommen, um die initialen Randmasken zu erzeugen. Um auszuschließen, dass man zur mittleren Strichlinie zugehörige Bildpunkte einer Randlinie zuordnet, wird zusätzlich eine elementweise UND-Operation mit der negierten Mittellinienmaske durchgeführt. Die verhinderte Überschneidung der Masken ist auch in Abbildung 7.1 zu erkennen.

#### 7.1.4. Vorteile von Ransac mit Maskierung

Eine kubische Funktion als mathematisches Modell der gesuchten Linie bietet einige Vorteile:

- Im Gegensatz zu einer Geraden, einem Kreis oder einer quadratischen Funktion kann das gewählte Polynom dritten Grades zusätzlich dem Verlauf einer S-Kurve (Übergang einer Links- in eine Rechtskurve und umgekehrt) relativ gut folgen.
- Unter der Annahme einer guten Annäherung zur echten Markierung kann durch

Extrapolation eine wahrscheinliche Vorhersage des Linienverlaufs in einem später aufgenommenen Bild getroffen werden.

- Die Stetigkeit der kubischen Funktion verhindert das plötzliche Abknicken des Linienverlaufs, sodass nahezu senkrecht auftreffende Querstraßenmarkierungen nicht als Kurve erkannt werden
- Durch die Anwendung von RANSAC haben Unterbrechungen oder kleine Störungen keinen negativen Einfluss. Als Outlier erkannte Pixel werden in der anschließenden Regression nicht berücksichtigt (s. Kapitel 4.5.1).

### 7.1.5. Probleme

Trotz der vielen Vorteile bringt gerade RANSAC das Problem mit sich, dass drei zu bestimmende, unabhängige Polynome in einem Bildausschnitt zu rechenintensiv und damit zeitaufwendig wären. Daher müssen in einer Aufnahme drei sinnvoll gewählte Bereiche feststehen, in welchen je ein RANSAC-Algorithmus ausgeführt werden kann. Das größte Problem, welches letztlich zum Ausschluss der RANSAC-Methode geführt hat, ist die zuverlässige Bestimmung dieser RoI. Ab Steigung von mehr als  $45^\circ$  approximiert das Polynom den Linienverlauf deutlich schlechter.

Entgegen der Erwartungen sind die Extrapolationseigenschaften von Polynomen zum Maskenbau ungeeignet. Da deren Wertebereich stets gegen unendlich läuft, weicht die Funktion am Bildrand meist stärker von der zu erkennenden Kurve ab. Dies führt zu falsch vorhergesagten RoIs, welche beispielhaft in Abb. 7.4 dargestellt sind.

Das Polynom hat keine Chance, die Fahrbahnmarkierung richtig abzubilden, da sie nicht zu den Punkten des maskierten Bereichs gehört. Jene fehlerhaften RoIs entstehen durch zuvor falsch erkannte, in die Weltkarte eingetragene Punkte. Abbildung 7.5 zeigt das Ergebnis eines Testlaufs. Dieser wurde mit Hilfe einer Aufnahme (rosbag<sup>1</sup>) durchgeführt, welche während manueller Steuerung des Fahrzeugs entstanden ist. Im dargestellten Plot der Weltkarte ist die anfänglich zufriedenstellende Detektion der Fahrbahnmarkierungen zu erkennen, welche jedoch ab der ersten Kreuzung verloren geht.

---

<sup>1</sup>Rosbag ist eine Reihe von Tools zum Aufnehmen und Wiedergeben von ROS-Topics ([wiki.ros.org/rosbag](http://wiki.ros.org/rosbag))

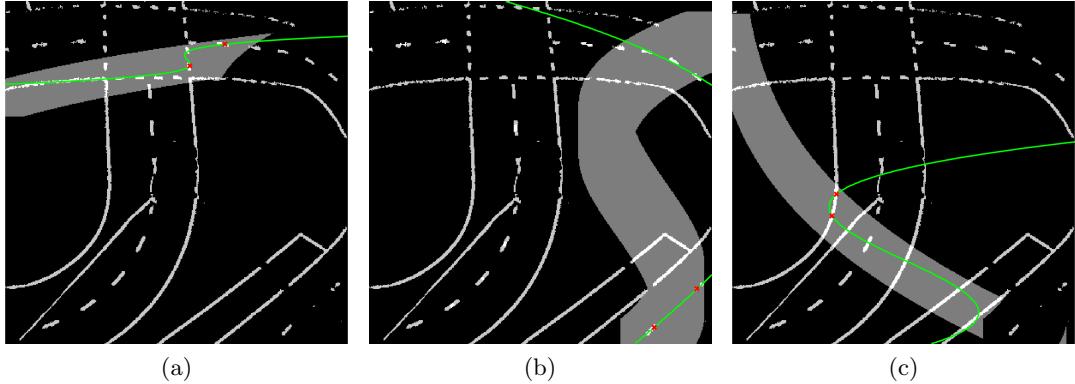


Abbildung 7.4.: Fehlerhaft erzeugte Masken der linken (a), mittleren (b) und rechten (c) Fahrbahnmarkierungen während eines Testlaufes. In Abb. 7.5 ist der Ausschnitt des Geschehens markiert

Wenn in einem zuletzt untersuchten Bild keine Linienpunkte gefunden wurden, muss die Maske des übernächsten Bildes aus den zuletzt eingetragenen Punkten erstellt werden. Diese Punkte können ungünstigerweise weit von der jetzigen Position entfernt liegen, sodass stark extrapoliert werden muss. In dieser Entfernung zu den Stützstellen bildet das die Maske beschreibende Polynom den wirklichen Linierverlauf ungenügend ab und RANSAC kann erneut keine richtigen Punkte finden. Somit ist die Voraussetzung gegeben, dass der Algorithmus im nächsten Foto erneut fehlschlägt, da falsche Punkte in der Karte zu noch stärkeren Abweichungen der Masken gegenüber der eigentlichen Markierung führen. Es entwickelt sich eine Kettenreaktion, die ein „Sich-Wieder-Fangen“ des Algorithmus sehr unwahrscheinlich macht. Hinzu kommt, dass selbst bei ideal ausgewählten RoIs die mathematische Beschreibung nicht jedem beliebigen Kurvenverlauf exakt folgen kann.

### 7.1.6. Fazit und Lösungsmöglichkeiten

Durch unser Paradigma, eine Weltkarte aufzubauen, ist der Gedanke naheliegend, deren Information zur Verarbeitung eines Bildes zu nutzen. Voraussetzung dafür ist allerdings, dass die Informationen der Karte hinreichend genau sind. Zugegebenermaßen könnte der RANSAC mit Maskierungen zu einem stabilen Verfahren ausgebaut werden, wenn eine Möglichkeit gefunden wird, die Bestimmung der ROI zuverlässiger zu gestalten.

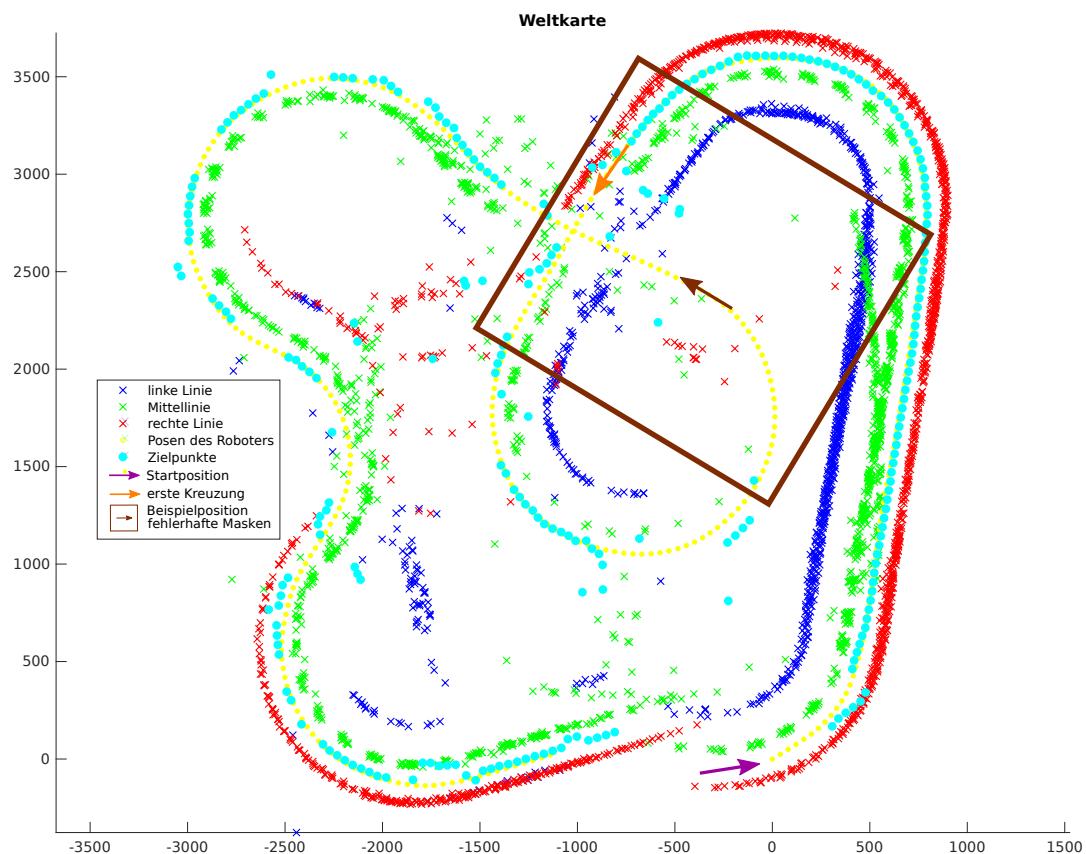


Abbildung 7.5.: Plot der Weltkarte nach einer Testrunde mittels manueller Steuerung. Die braune Umrundung bezeichnet das in Abbildung 7.4 dargestellte Sichtfeld

Beispielsweise wäre es denkbar, die maximale Krümmung des kubischen Polynoms zu beschränken, um so ein Abdriften der Masken an Kreuzungen zu unterbinden. Da ein Polynom jedoch stets ins Unendliche läuft, wäre möglicherweise eine andere mathematische Beschreibung sinnvoller gewesen. Wenigstens die Masken könnten aus zusammengesetzten Kreisen oder Splines hervorgehen. Außerdem könnte die Maskenbreite von der Entfernung bekannter Stützstellen abhängig gemacht werden, sodass in unbekannten Bereichen auf größerer Fläche gesucht wird. Zusätzlich wäre es möglich, RoIs über die bekannte Fahrspurbreite einander anzupassen und zu korrigieren, um ein Überlappen oder Vertauschen von linker und rechter Linie zu verhindern.

Falls im Rahmen weiterer Projekte der Riverflow-Algorithmus an seine Grenzen kommen sollte, ließe sich also unser erster Ansatz ebenfalls zum lauffähigen Programm erweitern. Die Riverflow-Methode bot allerdings die vielversprechende Aussicht auf eine robustere Funktionsweise ohne großartig aufwändige Anpassungen. Eine weitere Idee zur Verbesserung der Prädiktion einer Linie wird im nun folgenden Abschnitt zum Kalman-Filter beschrieben.

## 7.2. Kalman-Filter M

Auch die folgende Vorgehensweise zur Fahrspurerkennung approximiert den Verlauf der Straße als Polynom 3. Grades. Durch Veränderungen abseits dieser zentralen Komponente sollten jedoch Schwächen des 1. Ansatzes zur Erkennung der Fahrbahnmarkierungen behoben werden:

- Mittels eines prinzipbedingt als konstant angenommenen Abstand der rechten, mittleren und linken Linie sollte ein „Ineinanderlaufen“ der RoI zur Erkennung selbiger verhindert werden. Hierfür wird nicht mehr jede Fahrspurmarkierung einzeln, sondern nur noch die Mittellinie modelliert. Um die seitlichen Linien zu erhalten, kann dieser Verlauf verschoben werden.
- Das in Abschnitt 4.4 eingeführte Kalman-Filter soll verwendet werden, um die Lage der Fahrbahnmarkierungen in konsekutiven Aufnahmen zu verfolgen und somit zuverlässigere RoI Zur Erkennung der Linien aufzuspannen.
- Da nur kleine Teile des Bildes mit einem Kantendetektor bearbeitet werden und keine Binarisierung der Filterantwort stattfinden muss, sollte der Algorithmus schneller ablaufen können.

Final sollen die in Einzelbildern erkannten, mittig auf der Straßenmarkierung liegenden Punkte in die Weltkarte eingetragen und somit für den Regelungsalgorithmus verfügbar gemacht werden. Abbildung 7.6 vermittelt eine erste Idee der im Folgenden genauer beschriebenen Methode.

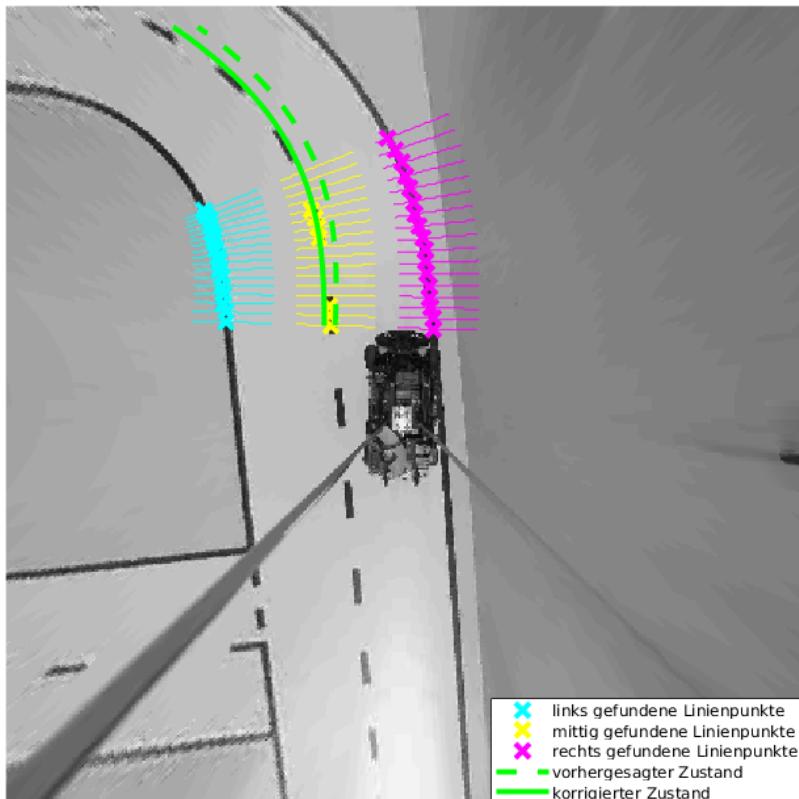


Abbildung 7.6.: Plot einer Iteration des Kalmanfilters

### 7.2.1. Fahrspurmodell

Als Repräsentation der Fahrspur im Zustandsraum soll das in (Peters, Falko 2009) vorgeschlagene Modell, welches wiederum eine Vereinfachung von (Risack u. a. 1998) darstellt, verwendet werden. Wie in bereits angekündigt, wird die Fahrspur als Polynom 3. Grades  $y^L(x^L)$  approximiert. Um die Systemmatrix  $\mathbf{A}$  zu vereinfachen und einen sehr anschaulichen Zustandsvektor  $\mathbf{x}$  zu erhalten, wird für jenen die 0. bis 3. Ableitung des Polynoms an der Stelle  $x^L = 0$  verwendet. Weitere Erläuterungen zu den

als Lateraler Versatz  $y_{off}$ , Gierwinkel  $\varphi$ , Fahrspurkrümmung  $c_0$  und Änderungsrate der Krümmung  $c_1$  interpretierten Komponenten des Zustandsvektors  $\boldsymbol{x}$  finden sich (Peters, Falko 2009, S. 47-48).

$$f(x^{\mathcal{L}}) = y^{\mathcal{L}}(x^{\mathcal{L}}) = a_0 + a_1 \cdot x^{\mathcal{L}} + a_2 \cdot (x^{\mathcal{L}})^2 + a_3 \cdot (x^{\mathcal{L}})^3 \quad (7.2)$$

$$\boldsymbol{x} = \begin{pmatrix} y^{\mathcal{L}}(0) \\ (y^{\mathcal{L}})'(0) \\ (y^{\mathcal{L}})''(0) \\ (y^{\mathcal{L}})'''(0) \end{pmatrix} = \begin{pmatrix} a_0 \\ a_1 \\ 2 \cdot a_2 \\ 6 \cdot a_3 \end{pmatrix} = \begin{pmatrix} y_{off} \\ \varphi \\ c_0 \\ c_1 \end{pmatrix} \quad (7.3)$$

$$y^{\mathcal{L}}(x^{\mathcal{L}}) = y_{off} + \varphi \cdot x^{\mathcal{L}} + \frac{1}{2} \cdot c_0 \cdot (x^{\mathcal{L}})^2 + \frac{1}{6} \cdot c_1 \cdot (x^{\mathcal{L}})^3 \quad (7.4)$$

### 7.2.1.1. Initialisierung

Die Initialisierung des Fahrspurmodells erfolgt durch das Finden der Mittellinie auf dem unmaskierten Bild wie in Abschnitt 7.1.3 oder mittels des Riverflow-Algorithmus für die mittlere Fahrbahnmarkierung, beschrieben in Passage 8.1. Sind alle der Mittellinie zugehörigen Bildpunkte gefunden, kann eine Transformation derer ins Linienkoordinatensystem  $\mathcal{L}$  mit anschließender Regression zum Feststellen der Parameter des Polynoms 3. Grades stattfinden. Mittels (7.3) lässt sich das gefundene Polynom in den Systemzustand  $\boldsymbol{x}$  überführen.

### 7.2.2. Messung

Um eine ausreichende Geschwindigkeit des Kalmanfilters zu gewährleisten, muss die Anzahl der Bildpunkte, welche zur Korrektur des prädizierten Zustands verwendet werden, im Gegensatz zu Abschnitt 7.1 vermindert werden. Es wird in Anlehnung an (Risack u. a. 1998) das Bild ab dem Vorverarbeitungsschritt nur in relevanten Bereichen betrachtet. Hierfür werden Geraden „Scanlines“ senkrecht zum Polynom des Fahrspurmodells berechnet (gelbe Linien in Abb. 7.6). Mathematisch lässt sich dies, basierend auf den Definitionsgleichungen (7.2) und (7.3) des Polynoms  $f(x^{\mathcal{L}}) = y^{\mathcal{L}}(x^{\mathcal{L}})$  und Zustandsvektors  $\boldsymbol{x}$  sowie den Formeln (7.1b) und (7.1c) zur Berechnung

des zum Polynom senkrechten, normierten Vektors  $\mathbf{d}$ , wie folgt ausdrücken:

$$\mathbf{m}(x^{\mathcal{L}}) = \begin{pmatrix} x^{\mathcal{L}} \\ \mathbf{f}(x^{\mathcal{L}}) \end{pmatrix} \quad (7.5a)$$

$$\mathbf{s}(x^{\mathcal{L}}, \alpha) = \mathbf{m}(x^{\mathcal{L}}) + \alpha \cdot \mathbf{d}(x^{\mathcal{L}}) \quad \alpha \in \mathbb{R} \quad (7.5b)$$

Die Länge der Scanlines (Wertebereich von  $\alpha$ ) sowie der Abstand der Startpunkte/Mittelpunkte  $\mathbf{m}(x^{\mathcal{L}})$  auf dem Polynom sind als Parameter anzusehen, welche Einfluss auf die RoI zum Erkennen der Mittellinie sowie die Anzahl der Messwerte haben.

Um mit der konstruierten Scanline Bildpunkte adressieren zu können, wird diese in eine diskrete Koordinatenserie überführt (7.6):

$$\mathbf{s}(x^{\mathcal{L}}, \alpha) = \mathbf{m}(x^{\mathcal{L}}) + \alpha \cdot \mathbf{d}(x^{\mathcal{L}}) \quad \alpha \in \mathbb{Z} \quad (7.6)$$

Anschließend müssen deren Elemente vom Linienkoordinatensystem  $\mathcal{L}$  ins Bildkoordinatensystem  $\mathcal{I}$  transformiert und auf ganzzahlige Werte gerundet werden. Die durch die Koordinatenserie adressierten Pixel des entzerrten Bildes können nun zur weiteren Verarbeitung in einen Zeilenvektor  $\mathbf{l}$  geschrieben werden.

$$\mathbf{l} = \begin{pmatrix} l_1 & l_2 & \dots & l_i & \dots & l_n \end{pmatrix} \quad (7.7)$$

Nun wird  $\mathbf{l}$  mit dem aus Passage 6.2 bekannten Kantendetektor gefiltert. Von der entstandenen Filterantwort  $\mathbf{f}$  wird das Maximum  $\max(\mathbf{f})$  ermittelt. Ist  $\max(\mathbf{f})$  größer als ein bestimmter Schwellwert, wird der Index  $i_{max}$  des Maxima in  $\mathbf{f}$  vormerkt. Um eine Linie nicht wiederholt zu finden, muss die in  $\mathbf{f}$  verbleibende, durch dieselbe Markierung verursachte Filterantwort entfernt werden (7.8):

$$l_{i_{max}-\text{Linienbreite}/2} \dots l_{i_{max}} \dots l_{i_{max}+\text{Linienbreite}/2} = 0 \quad (7.8)$$

Darauffolgend kann nach weiteren, den festgelegten Grenzwert überschreitenden Maxima in  $\mathbf{f}$  gesucht und wie mit dem ersten  $\max(\mathbf{f})$  verfahren werden. Wird kein ausreichend großes  $\max(\mathbf{f})$  mehr gefunden, können den gefundenen Indizes  $i_{max}$  via (7.6) und einer Rücktransformation vom Bildkoordinatensystem  $\mathcal{I}$  Punkte  $\mathbf{p}^{\mathcal{L}}$  im Linienkoordinatensystem  $\mathcal{L}$  zugeordnet werden, welche zur Korrektur des Systemzustands  $\mathbf{x}$  vorgesehen sind.

### 7.2.2.1. Randlinien

Um auch die zentral in der äußereren Fahrbahnmarkierung liegenden Punkte detektieren zu können, werden die Scanlines der Mittelline um die Fahrspurbreite senkrecht zum Polynom verschoben (cyanfarbene und pinke Linien in Abb. 7.6). Nun kann der selbe Algorithmus wie für die Scanlines der mittleren Fahrbahnmarkierung ausgeführt werden. Die gefundenen Koordinaten müssen letztendlich wieder um die Fahrspurbreite zur Mittellinie hin verschoben werden um problemlos zur Zustandskorrektur des Kalmanfilters nutzbar zu sein.

### 7.2.3. Zustandsraumbeschreibung

Bei Nutzung eines Kalmanfilters zur Extraktion von Fahrspurinformationen aus konsekutiven Bildern benötigt man wie in Abschnitt 4.4.1 angekündigt ein Modell, welches die Lage der Straße im Folgebild vorhersagen kann. Eine ausführliche Herleitung der nun nur kurz umrissenen, da final nicht verwendeten Zustandsraumbeschreibung findet sich in (Peters, Falko 2009, S. 47-50).

#### 7.2.3.1. Herleitung der Systemmatrix

Die Systemmatrix  $\mathbf{A}$  bildet den momentanen Zustandsvektor  $\mathbf{x}(k)$  auf den folgenden Zustandsvektor  $\mathbf{x}(k+1)$  ab, ohne äußere Einflüsse auf das System zu berücksichtigen. Die Lage der Fahrbahnmarkierungen im nächsten Bild soll also auf Basis des aktuellen Bildes vorhergesagt werden. Unter Vernachlässigung des Lenkwinkels wie in (Peters, Falko 2009, S. 48) kann der Folgezustand  $\mathbf{x}(k+1)$  nur auf Basis des zwischen zwei Bildaufnahmepunkten zurückgelegten Weges  $\Delta x_{\mathcal{L}}$  berechnet werden. Da sich der Roboter ohne Lenkeinschlag geradlinig in Richtung der Abszisse des Fahrspurkoordinatensystems  $\mathcal{L}$  bewegt, muss lediglich eine Auswertung des Polynoms und seiner Ableitungen an der Stelle  $\Delta x_{\mathcal{L}}$  erfolgen, um den konsekutiven Zustandsvektor zu erhalten:

$$\hat{\mathbf{x}}(k+1) = \begin{pmatrix} y^{\mathcal{L}k}(\Delta x^{\mathcal{L}}) \\ (y_k^{\mathcal{L}})'(\Delta x^{\mathcal{L}}) \\ (y_k^{\mathcal{L}})''(\Delta x^{\mathcal{L}}) \\ (y_k^{\mathcal{L}})'''(\Delta x^{\mathcal{L}}) \end{pmatrix} \quad (7.9)$$

In (Peters, Falko 2009) wird beschrieben, dass sich aufgrund von Nichtlinearität

aus (7.9) keine äquivalente Matrix  $\mathbf{A}$  aufstellen lässt. Da diese Nichtlinearität jedoch nur in Bezug auf  $\Delta x$  und nicht den Zustandsvektor  $\mathbf{x}$  besteht, ist dies trotz dessen möglich. Aus (7.4) und (7.9) folgt:

$$\hat{\mathbf{x}}(k+1) = \begin{pmatrix} y_{off} + \varphi \cdot \Delta x + \frac{1}{2} \cdot c_0 \cdot (\Delta x)^2 + \frac{1}{6} \cdot c_1 \cdot (\Delta x)^3 \\ \varphi + c_0 \cdot \Delta x + \frac{1}{2} \cdot c_1 \cdot (\Delta x)^2 \\ c_0 + c_1 \cdot \Delta x \\ c_1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & \Delta x & \frac{1}{2} \cdot (\Delta x)^2 & \frac{1}{6} \cdot (\Delta x)^3 \\ 0 & 1 & \Delta x & \frac{1}{2} \cdot (\Delta x)^2 \\ 0 & 0 & 1 & \Delta x \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} y_{off} \\ \varphi \\ c_0 \\ c_1 \end{pmatrix} \quad (7.10)$$

$$\mathbf{A} \cdot \begin{pmatrix} y_{off} \\ \varphi \\ c_0 \\ c_1 \end{pmatrix}$$

Die in (Peters, Falko 2009) durch Linearisierung an der Stelle  $x^{\mathcal{L}0} = 0$  erhaltene Matrix hingegen vereinfacht sich durch (7.12) zu:

$$\hat{\mathbf{x}}(k+1) = \begin{pmatrix} y_{off} + \varphi \cdot \Delta x \\ \varphi + c_0 \cdot \Delta x \\ c_0 + c_1 \Delta x \\ c_1 \end{pmatrix} = \begin{pmatrix} 1 & \Delta x & 0 & 0 \\ 0 & 1 & \Delta x & 0 \\ 0 & 0 & 1 & \Delta x \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} y_{off} \\ \varphi \\ c_0 \\ c_1 \end{pmatrix} = \mathbf{A} \cdot \begin{pmatrix} y_{off} \\ \varphi \\ c_0 \\ c_1 \end{pmatrix} \quad (7.11)$$

$$f(x^{\mathcal{L}}) \approx f(x^{\mathcal{L}0}) + f'|_{x^{\mathcal{L}0}} \cdot (x - x_0) \quad (7.12)$$

Da sich die gefahrene Strecke  $\Delta x_{\mathcal{L}}$  zwischen den Aufnahmepunkten zweier Bilder geringfügig ändern kann, muss die Systemmatrix  $\mathbf{A}$  vor jedem Prädikationsschritt neu berechnet werden.

### 7.2.3.2. Ausgangsmatrix/Messmatrix

Die Ausgangsmatrix/Messmatrix  $\mathbf{C}$  bildet den Systemzustand  $\mathbf{x}$  auf eine Messung  $\mathbf{y}$  ab. In unserer Implementierung wird  $\mathbf{C}$  passend zu den  $x$ -Koordinaten der wie in Abschnitt 7.2.2 beschrieben gewonnenen Linienmittelpunkte in jeder Iteration des Kalmanfilters neu berechnet. Aus (7.4) ergibt sich:

$$\begin{aligned} \mathbf{y} &= \begin{pmatrix} y_1^{\mathcal{L}} \\ y_2^{\mathcal{L}} \\ \vdots \\ y_n^{\mathcal{L}} \end{pmatrix} = \begin{pmatrix} y_{off} + \varphi \cdot x_1^{\mathcal{L}} + \frac{1}{2} \cdot c_0 \cdot (x_1^{\mathcal{L}})^2 + \frac{1}{6} \cdot c_1 \cdot (x_1^{\mathcal{L}})^3 \\ y_{off} + \varphi \cdot x_2^{\mathcal{L}} + \frac{1}{2} \cdot c_0 \cdot (x_2^{\mathcal{L}})^2 + \frac{1}{6} \cdot c_1 \cdot (x_2^{\mathcal{L}})^3 \\ \vdots \\ y_{off} + \varphi \cdot x_n^{\mathcal{L}} + \frac{1}{2} \cdot c_0 \cdot (x_n^{\mathcal{L}})^2 + \frac{1}{6} \cdot c_1 \cdot (x_n^{\mathcal{L}})^3 \end{pmatrix} \quad (7.13) \\ &= \begin{pmatrix} 1 & x_1^{\mathcal{L}} & \frac{1}{2} \cdot (x_1^{\mathcal{L}})^2 & \frac{1}{6} \cdot (x_1^{\mathcal{L}})^3 \\ 1 & x_2^{\mathcal{L}} & \frac{1}{2} \cdot (x_2^{\mathcal{L}})^2 & \frac{1}{6} \cdot (x_2^{\mathcal{L}})^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_n^{\mathcal{L}} & \frac{1}{2} \cdot (x_n^{\mathcal{L}})^2 & \frac{1}{6} \cdot (x_n^{\mathcal{L}})^3 \end{pmatrix} \cdot \begin{pmatrix} y_{off} \\ \varphi \\ c_0 \\ c_1 \end{pmatrix} \end{aligned}$$

### 7.2.4. Kovarianzmatritzen

Wie in (Peters, Falko 2009) und (Risack u.a. 1998) werden die Kovarianzmatrix des System-/Prozessrauschen  $\mathbf{Q}$  und die Kovarianzmatrix des Messrauschen  $\mathbf{R}$  als konstante Diagonalmatritzen gesetzt. Auf die Prädiktion der Kovarianzmatrix  $\mathbf{P}$  wird somit verzichtet.

### 7.2.5. konkrete Implementierung des Kalman-Filters

Ist der Zustandsvektor  $\mathbf{x}$  einmal wie in Passage 7.2.1.1 beschrieben initialisiert, können die Gleichungen des Kalmanfilters bei Eintreffen eines neuen Bildes wie folgt berechnet werden:

1. Bildung der Systemmatrix  $\mathbf{A}$  mittels der seit dem letzten Bild gefahrenen Distanz  $\Delta x$
2. Prädiktion des Zustandsvektor  $\mathbf{x}$ , d.h. der Lage der Fahrspurmarkierungen im aktuellen Bild anhand von (4.16)
3. Messung der Lage der Fahrspur wie in Abschnitt 7.2.2 erläutert

4. Bildung der Ausgangsmatrix/Messmatrix  $\mathbf{C}$  passend zur Messung wie in Passage 7.2.3.2 beschrieben
5. Korrektur des Zustandsvektors  $\mathbf{x}$  unter Nutzung von (4.15)

### 7.2.6. Vorteile

Durch den Einsatz der Scanlines anstatt einer Verwendung des RANSAC-Algorithmus im Zusammenhang mit Maskierung der entsprechenden Bildausschnitte läuft die Bildverarbeitung ca. um Faktor 5 schneller ab.

### 7.2.7. Probleme

Die Vorhersagen des Folgezustands durch das Zustandsraummodell (7.10) sind oft so schlecht, dass Scanlines Punkte einer falschen Fahrbahnmarkierung erkennen. Somit „wandert“ das im Zustandsvektor  $\mathbf{x}$  repräsentierte Polynom durch die fehlerhafte Korrektur von der Mittellinie, welche es darstellen soll, zu einer der Randlinien und schließlich völlig abseits der Straße. Ein Beispiel stellt die ungenügende Vorhersage des Fahrspurpolynoms in Abb. 7.7c dar, hier werden Punkte der linken Fahrbahnmarkierung als Mittellinie detektiert. Durch die daraus folgende fehlerhafte Korrektur und die somit noch schlechtere Vorhersage in der nächsten Iteration kommt es in 7.7d sogar zu einer Detektion der linken Außenlinie als rechte Fahrbahnmarkierung.

Eine Variante dieses Phänomen zu verhindern, wäre eine Verbesserung des Zustandsraummodells. Ein Einbinden des Lenkwinkels, der Winkelgeschwindigkeit oder Orientierung des Fahrzeugs als Eingang  $\mathbf{u}$  wäre theoretisch möglich. Da der Zusammenhang zwischen diesen Größen und den Elementen des Zustandsvektors  $\mathbf{x}$  leider nicht trivial ist und keine Literaturquelle vorliegt, in der ein Zustandsraummodell mit diesen Eingangsgrößen gezeigt ist, wurde jedoch vorerst darauf verzichtet. Oft wird in der Literatur bei Nutzung eines Kalmanfilters zur Fahrspurverfolgung sogar gänzlich von einer Vorhersage abgesehen, d.h.  $\mathbf{A} = \mathbf{I}$  (Lim, Seng und Ang 2012).

Eine weitere Möglichkeit zur Verbesserung der Performance des vorgeschlagenen Algorithmus stellt die Erhöhung der Bildrate dar. Hierdurch kann eine ungenaue Vorhersage durch häufige Korrektur ausgeglichen werden. Durch eine noch effizientere Implementierung, Parameteroptimierung sowie eine Verkleinerung der Auflösung und/oder des Bildausschnittes könnte dies erzielt werden. Da der in Passage 8 beschriebene Ansatz ohne jegliche Optimierung im jetzigen Testszenario weitaus bessere

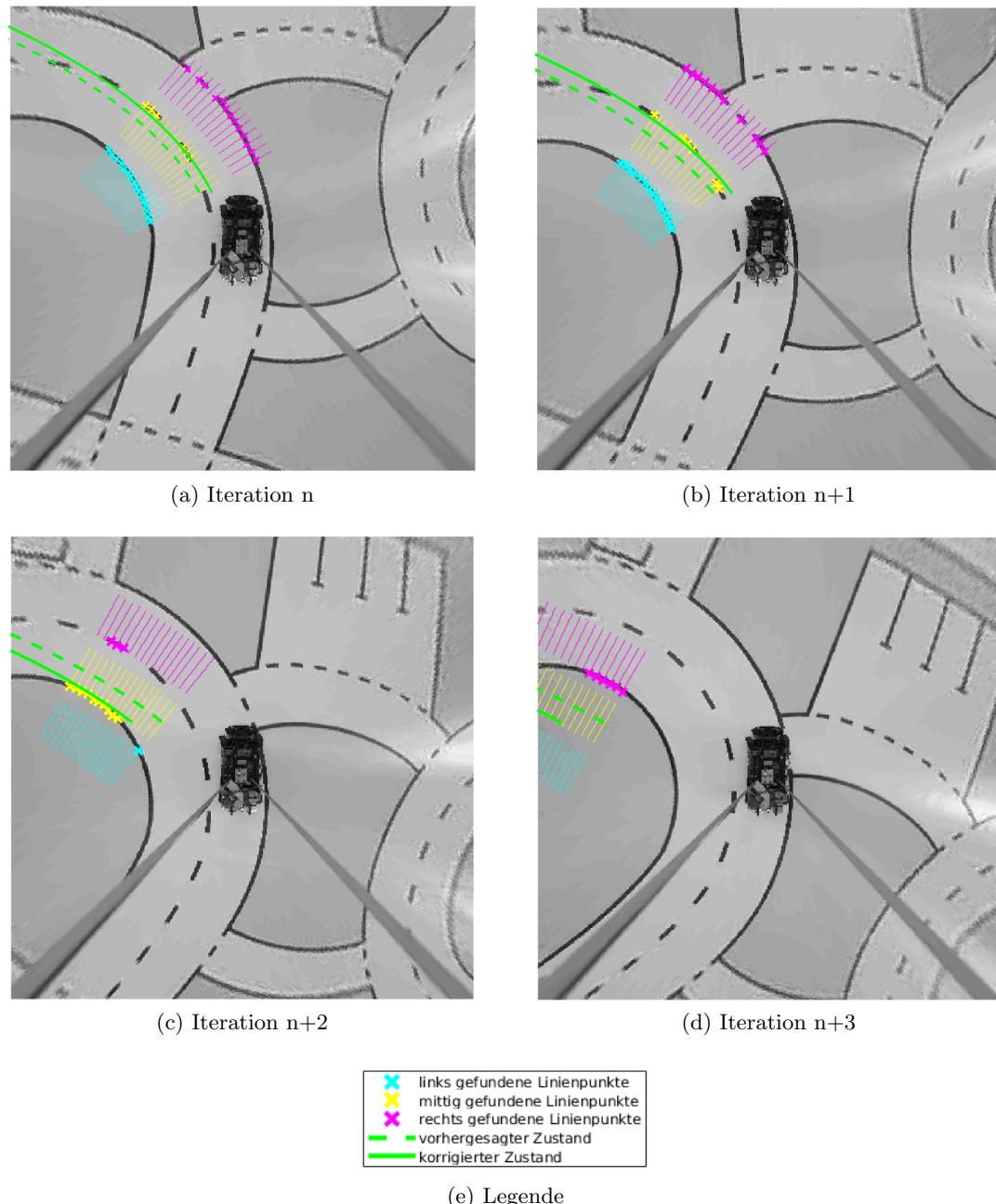


Abbildung 7.7.: „Weglaufen“ des Kalmanfilter-Zustands durch ungenügende Vorhersage

Es werden aufeinanderfolgende Iterationen des Kalmanfilters ( $n$  bis  $n+3$ ) gezeigt. Neben den in der Legende vorhandenen Einträgen sind die zur Linienpunkt-Detektion genutzten Scanlines als dünne Geraden vermerkt.

---

Ergebnisse lieferte, wurde jedoch keine weitere Verfeinerung dieser Methode vorgenommen.

Einen weiteren signifikanten Nachteil stellt wie in Abschnitt 7.1.5 beschrieben die Modellierung der Fahrspur durch ein Polynom 3. Grades  $y(x)$  dar, welches schon eine  $90^\circ$ -Kurve sehr schlecht approximieren kann, da dessen Anstieg unendlich werden müsste. Die volle Sichtweite des Fahrzeugs kann somit nicht genutzt werden (siehe z.B. Abb. 7.7).

## 8. Punktbasierte Fahrspurerkennung M

Der dritte und final genutzte Ansatz zur Fahrspurerkennung nutzt im Gegensatz zu den vorherigen Varianten kein durch wenige Parameter (Polynomkoeffizienten etc.) beschreibbares Modell. Dem Objekt Fahrspur werden hingegen bestimmte Eigenschaften zugesprochen, welche zur Idee der punktbasierten Detektion und angeschlossenen Verifikation geführt haben (Idee des Regelbasierten Ansatzes entnommen aus (Drauschke 2016, S. 35-39)).

1. Ist eine Fahrbahnmarkierung eine Randlinie, so ist dies eine nicht unterbrochene Markierung bestimmter Breite.
2. Ist eine Fahrbahnmarkierung eine Mittellinie, so besteht diese aus Elementen deren Länge, Breite und Abstand voneinander konstant und bekannt sind.
3. Die Krümmung einer Fahrbahnmarkierung darf einen bestimmten Wert nicht überschreiten.
4. Rechte-, Mittel- und Seitenlinie besitzen einen konstanten Abstand (Fahrspurbreite) zueinander.

In den folgenden Abschnitten wird gezeigt, wie mittels der Eigenschaft 2 unter Nutzung der MATLAB-Funktion *regionprops* mögliche Mittellinienelemente aus dem Binärbild extrahiert werden. Darauf folgt die Erläuterung einer iterativen Methode zur Extraktion für die seitliche Fahrbahnmarkierung repräsentativer Punkte auf Basis der Charakteristika 1 und 3 („Riverflow-Algorithmus“). Anschließend wird auf das Konzept zur Selektion der richtig erkannten und für die Fahrspurverfolgung relevanten Mittellinienelemente aus allen Ergebnissen der *regionprops*-Funktion eingegangen, welche an das Verfahren zur Detektion der Randlinien angelehnt ist. Da bei der Erkennung einer seitlichen Fahrbahnmarkierung mehrere mögliche Verläufe („Hypothesen“) generiert werden können, sowie eine Überprüfung der Plausibilität

der erkannten Koordinatenserien(n) wünschenswert ist, erfolgt basierend auf dem in Spezifikum 4 erwähnten Modellwissen der Straße ein Vergleich des erkannten Verlaufs / der erkannten Verläufe mit der gegenüberliegenden Rand- sowie Mittellinie. Eine Auswahl der längsten zu den übrigen extrahierten Punkten passenden Hypothese kann somit stattfinden. Abschließend wird auf den Sonderfall einer unterbrochenen seitlichen Fahrbahnmarkierung eingegangen, dessen Bearbeitung sich aus einer Fusion der für die Mittel- und Randlinie implementierten Verfahren ergibt.

## 8.1. Mittellinie K

Ähnlich wie in Kapitel 7.1 beschrieben findet dieser Ansatz die zur Mittellinie zugehörigen Elemente, ohne auf eine richtig definierte RoI angewiesen zu sein. Ausgangspunkt ist ebenfalls das binarisierte Bild, auf das die MATLAB-Funktion *regionprops* angewendet wird. Diese bereits verfügbare Funktion findet als Alternative zum in Abschnitt 7.1 erwähnten „Ringfilter“ alle zusammenhängende Pixelgruppen und extrahiert zusätzlich eine Vielzahl derer Eigenschaften, wie Länge, Breite, Orientierung und Mittelpunkt. Da wie in Punkt 2 bereits erwähnt die mittlere Fahrbahnmarkierung genau spezifiziert ist, werden so alle potentiellen Mittellinienstriche vorgemerkt. Diese Vorgehensweise funktioniert allerdings nur dann stabil, wenn die Beschaffenheit des binarisierten Bildes gut genug ist, sodass z.B. Striche nicht unterbrochen sind. Trotz der guten Qualität des Bildes kommt es vor, dass falsche Bildteile als Mittellinienelemente erkannt und vorgemerkt werden. Deswegen stellen wir durch eine Verifikation (siehe Abschnitt 8.3.1) sicher, dass möglichst keine falschen Punkte in die Weltkarte, welche später zur Regelung des Autos genutzt wird, eingetragen werden. Schon an der großen Vier-Seiten-Kreuzung kommt es sonst zum Problem, dass darüber hinaus Mittellinienstriche der kreuzenden Straße erkannt werden. Auch wenn diese zur mittleren Linie gehören, sollen sie hier auf Grund des derzeitigen Regelungskonzepts noch nicht in die Karte aufgenommen werden.

## 8.2. Randlinie

Dieser Abschnitt erklärt den final zur Detektion der Randlinien genutzten Riverflow-Algorithmus. Er besteht aus den wesentlichen drei Schritten der Startpunktgewinnung, dem Finden von Markierungspunkten mittels Scanlinien und der abschließenden

den Verifikation der gefundenen Punkte.

### 8.2.1. Startpunktgewinnung M

Um den eigentlichen Riverflow-Algorithmus ausführen zu können, wird die ungefähre Lage der seitlichen Fahrbahnmarkierungen in der Nähe des Fahrzeugs benötigt. Zur Ermittlung dieser gibt es 3 Möglichkeiten:

1. Die Bestimmung durch Orientierung und Position des ersten vor dem Fahrzeug gelegenen Mittellinienelementes. Hierzu wird der Mittelpunkt des Elementes senkrecht zu seiner Orientierung um die Fahrspurbreite nach links/rechts verschoben (siehe Abb. 8.1).
2. Eine eindimensionale Hough-Transformation des Bildausschnittes direkt vor dem Fahrzeug wie in Abschnitt 4.6.1 beschrieben (siehe Abb. 8.2).
3. Annahme einer festen Position vor dem Fahrzeug/im Kamerabild.

Die Verfahren 2 oder 3 werden nur genutzt, wenn die Herangehensweisen 1 bzw. 1 und 2 kein Ergebnis lieferten.

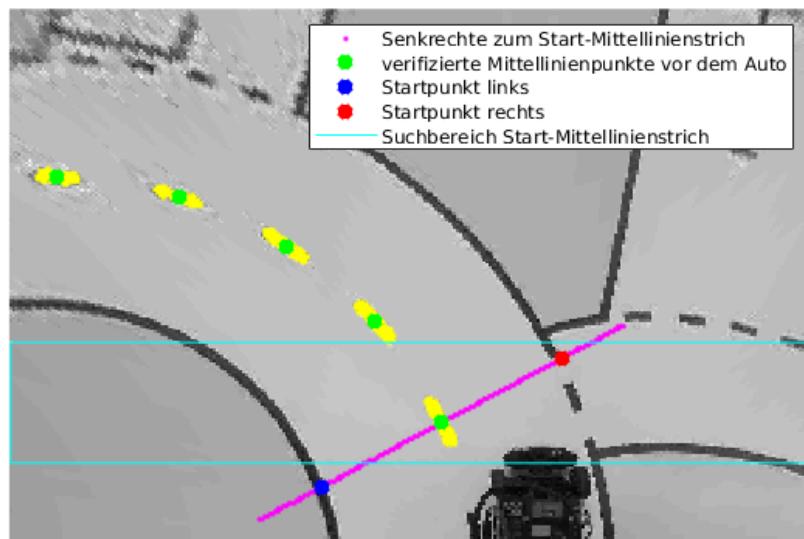


Abbildung 8.1.: Startpunktdefinition des Riverflow-Algorithmus für die Randlinien anhand der mittleren Fahrbahnmarkierung

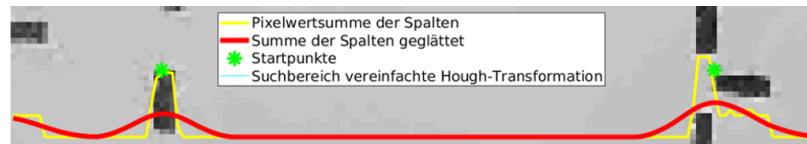


Abbildung 8.2.: Startpunktdefinition des Riverflow-Algorithmus für die Randlinien durch eindimensionale Hough-Transformation

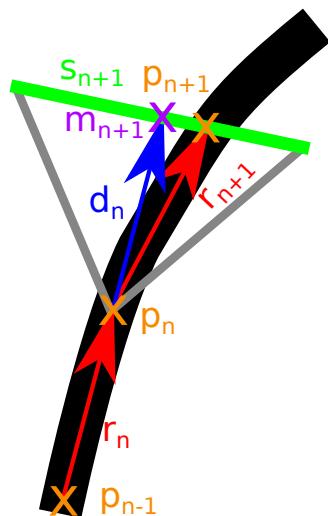


Abbildung 8.3.: Funktionsweise des Riverflow-Algorithmus

### 8.2.2. zentraler Algorithmus

Die Idee zum im folgenden Abschnitt dargelegten Ansatz zur Verfolgung durchgängiger Fahrbahnmarkierungen entstand auf Basis von (Drauschke 2016) sowie (Lim, Seng und Ang 2012). Durch Nutzung der Eigenschaften 1 und 3 kann ausgehend von der aktuellen Linienorientierung in einem bestimmten Kegel (graue Linien in Abb. 8.3), dessen Öffnungswinkel durch die maximale Krümmung der Fahrspur definiert ist, der weitere Verlauf der Fahrbahnmarkierung angenommen werden. Algorithmisch genutzt wird dieses Wissen durch die Verwendung der aus Abschnitt 7.2.2 bekannten Scanlines. Ausgehend vom letzten gefundenen Punkt der Linie  $\mathbf{p}_n$  wird selbiger um den Vektor  $\mathbf{d}_n$  weiterverschoben und bildet den Mittelpunkt  $\mathbf{m}_{n+1}$  der nächsten Scanline (8.2). Der Verschiebungsvektor  $\mathbf{d}_n$  wird auf Basis des vorletzten ( $\mathbf{p}_{n-1}$ ) und letzten gefundenen Punktes ( $\mathbf{p}_n$ ) der Fahrbahnmarkierung gebildet (8.1). Er stellt den normierten, mit der gewünschten Verschiebungslänge  $l_v$  multiplizierten Vektor zwischen ( $\mathbf{p}_{n-1}$ ) und ( $\mathbf{p}_n$ ) dar.

$$\mathbf{d}_n = \frac{\mathbf{p}_n - \mathbf{p}_{n-1}}{\|\mathbf{p}_n - \mathbf{p}_{n-1}\|} \cdot l_v = \begin{pmatrix} d_{nx} \\ d_{ny} \end{pmatrix} \quad (8.1)$$

$$\mathbf{m}_{n+1} = \mathbf{p}_n + \mathbf{d}_n \quad (8.2)$$

Mithilfe des zu  $\mathbf{d}_n$  (8.1) senkrechten Richtungsvektors  $\mathbf{l}_{n+1}$  (8.4) der ( $n + 1$ )-sten Scanline  $\mathbf{s}_{n+1}$  kann selbige wie folgt beschrieben werden:

$$\mathbf{s}_{n+1} = \mathbf{m}_{n+1} + \alpha \cdot \mathbf{l}_{n+1} \quad \alpha \in \mathbb{R} \quad (8.3)$$

$$\mathbf{l}_{n+1} = \begin{pmatrix} -d_{ny} \\ d_{nx} \end{pmatrix} \quad (8.4)$$

Der Wertebereich des skalaren Faktors  $\alpha$  gibt hierbei die Ausdehnung der Scanline um ihren Startpunkt/Mittelpunkt  $\mathbf{m}_{n+1}$  herum an.

Da die Filterantwort des Kantendetektors für das gesamte Bild bereits vorliegt, um die Mittellinie detektieren zu können, kann auch zur Erkennung der seitlichen Fahrbahnmarkierung auf diese zurückgegriffen werden. Bei der nun folgenden, in Abschnitt 7.2.2 ausführlich beschriebenen Extraktion von zentral auf der Fahrbahnmarkierung liegenden Punkten, entfällt also die Notwendigkeit der Filterung. Da

der Riverflow-Algorithmus sich im Bildkoordinatensystem  $\mathcal{I}$  bewegt, ist auch die in Passage 7.2.2 notwendige Hin- und Rücktransformation vom bzw. zum Linienkoordinatensystem  $\mathcal{L}$  nicht notwendig. In allen anderen Punkten gleicht der Weg von der aufgestellten Scanline  $s$  zu den darauf ermittelten, zentral in der Fahrbahnmarkierung liegenden Koordinaten  $p$  dem in Abschnitt 7.2.2 dargelegten.

Sind mehrere auf der Scanline  $s_{n+1}$  gefundene Punkte  $p_{n+1}$  vorhanden, so werden ab diesem Zeitpunkt multiple Hypothesen einer seitlichen Fahrbahnmarkierung verfolgt. Jetzt kann die nächste Iteration des Riverflow-Algorithmus mit der Bildung des folgenden Verschiebungsvektors bzw. der folgenden Verschiebungsvektoren  $d_{n+1}$  starten.

### 8.2.3. Sonderfall 1. und 2. Iteration

Da in der 1. und 2. Iteration keine Punkte  $p_{n-1}$  und  $p_n$  vorhanden sind, muss der Verschiebungsvektor  $d$  anderweitig bestimmt werden. Wurde der Startpunkt wie in Methode 1 beschrieben durch ein Mittellinienelement bestimmt, kann dessen Orientierung als Richtung des Verschiebungsvektors genutzt werden. Bei den Verfahren 2 und 3 wird eine konstante Verschiebung in Fahrtrichtung des Fahrzeugs genutzt.

### 8.2.4. Ende des Algorithmus

Der Algorithmus wird beendet, sobald:

1. Der Mittelpunkt  $m$  der nächsten Scanline außerhalb des Bildbereichs liegt
2. Auf der aktuellen Scanline keine Maxima größer als der Schwellwert  $s$  gefunden wurden. Vor dem endgültigem Abbruch der Verfolgung einer Fahrbahnmarkierung wird der Verschiebungsvektor  $d$  der ergebnislosen Scanline verlängert und die erneute Suche nach einer ausreichend großen Filterantwort durchgeführt, somit können auch kleine Unterbrechungen einer Linie überwunden werden.
3. Die betragsmäßige Orientierungsänderung  $|\Delta\phi| = |\text{atan}2(r_{n+1y}, r_{n+1x}) - \text{atan}2(r_{ny}, r_{nx})|$  der zwei Vektoren  $r_{n+1} = p_{n+1} - p_n$  sowie  $r_n = p_n - p_{n-1}$  zwischen den letzten zwei gespeicherten und dem gerade gefundenen Punkt einen Grenzwert nicht überschreitet (s. Abb. 8.3). Da der Straßenverlauf im

gegebenen Szenario niemals abknickt, wäre dies ein sicheres Zeichen einer Fehlerkennung. Diese Abbruchbedingung ließe sich auch durch Die Länge der Scanlines einstellen, indem somit der Öffnungswinkel des Suchkegels (graue Linien in Abb. 8.3) reduziert wird und Punkte, die zu einem Abknicken der erkannten Fahrbahnmarkierung führen, gar nicht erst erkannt werden. Die Prüfung der Orientierungsänderung wird jedoch beim in Abschnitt 8.4 beschriebenen Sonderfall der Extraktion des Punktes  $p_{n+1}$  notwendig.

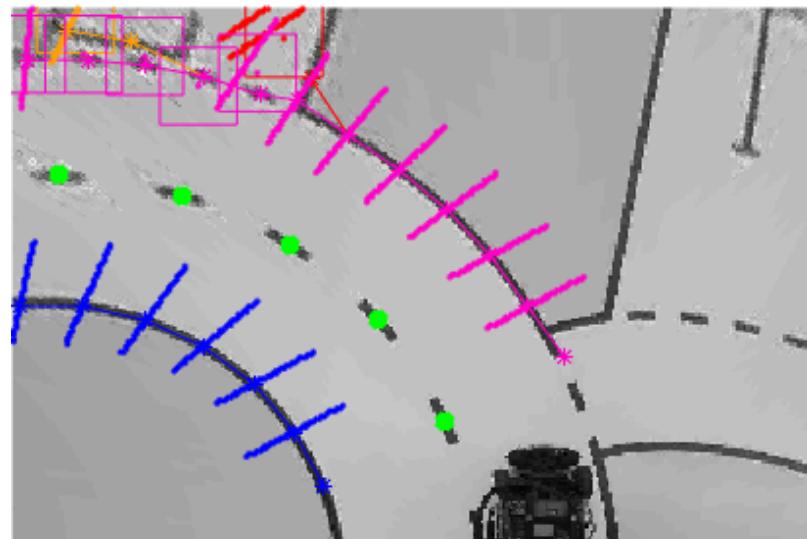


Abbildung 8.4.: Plot eines Riverflow-Durchlaufs für die Randlinien:

Die Farben Rot, Orange und Pink stellen die Hypothesen der rechten, Blau die einzige Hypothese der linken Fahrbahnmarkierung dar. Diese Hypothesen bestehen jeweils aus den \* für gefundene Punkte, dünnen Linien für deren Verschiebungsvektoren zueinander, fettgedruckten Geraden zur Darstellung der Scanlines sowie Scanquadra-ten und deren Mittelpunkte (.) im Bereich der unterbrochenen Mar-kierung. Die grünen Punkte repräsentieren die bereits erkannten & verifizierten Mittellinienpunkte.

## 8.3. Verifikation M

Nachdem für Mittel- und Seitenlinie repräsentative Punkte aus dem Bild extrahiert wurden, gilt es deren Sinnhaftigkeit zu überprüfen. Somit können:

1. Falsch erkannte Fahrbahnmarkierungselemente eliminiert werden.
2. Bei Vorhandensein mehrerer Hypothesen für die Seitenlinien die Besten ausgewählt werden.

### 8.3.1. Mittellinie K

Die Prozedur der Verifizierung verläuft nach einem vergleichbaren Prinzip wie der in Abschnitt 8.2 beschriebene Riverflow-Algorithmus. Dabei wird davon ausgegangen, dass sich das Fahrzeug größtenteils in der Fahrspur befindet, sodass mindestens ein Mittellinienstrich unweit vor dem Auto positioniert ist. Initial wird ein Suchfenster erstellt, welches in der Höhe den reichlichen Abstand zwischen den Mittelpunkten zweier benachbarter Mittelstreifen annimmt. Das erste sich in diesem Bereich aufhaltende Objekt wird dann als verifiziert angesehen, wenn seine Orientierung nicht übermäßig von der des Fahrzeugs abweicht. Ausgehend von diesem Punkt „hangelt“ sich der Verifikations-Algorithmus von Strich zu Strich, indem er eine neue RoI (ein neues Suchfenster) aufstellt. Da zu jedem (verifizierten) Punkt seine Orientierung bekannt ist, wird der nächste Mittellinienpunkt in dieser Richtung im ebenfalls bekannten Abstand zwischen zwei Punkten vermutet und dort der Mittelpunkt des neuen Suchfensters definiert. Dieses Schema wiederholt sich, bis entweder der Bildrand erreicht ist, keine Punkte im Suchfenster liegen, oder sich die Orientierung von einem zum nächsten Strich zu stark geändert hat.

Das gleiche Vorgehen wird zudem wahlweise vom ersten Punkt hinter dem Fahrzeug ausgehend wiederholt, sodass auch schon passierte Mittellinienpunkte, welche z.B. bei einem zukünftig implementierten Überholvorgang benötigt werden könnten, verifiziert sind.

In Abb. 8.5 sehen wir das Resultat der Mittellinien-/Strichererkennung. Im gleichen Zug wird die *regionprops*-Funktion auch zur Randstrichererkennung bei Kreuzungen genutzt. Die Verarbeitung dieser Informationen kommt im Riverflow-Algorithmus während des Sonderfalls einer unterbrochenen Randlinie zum Einsatz, welcher im Abschnitt 8.4 näher beschrieben ist.

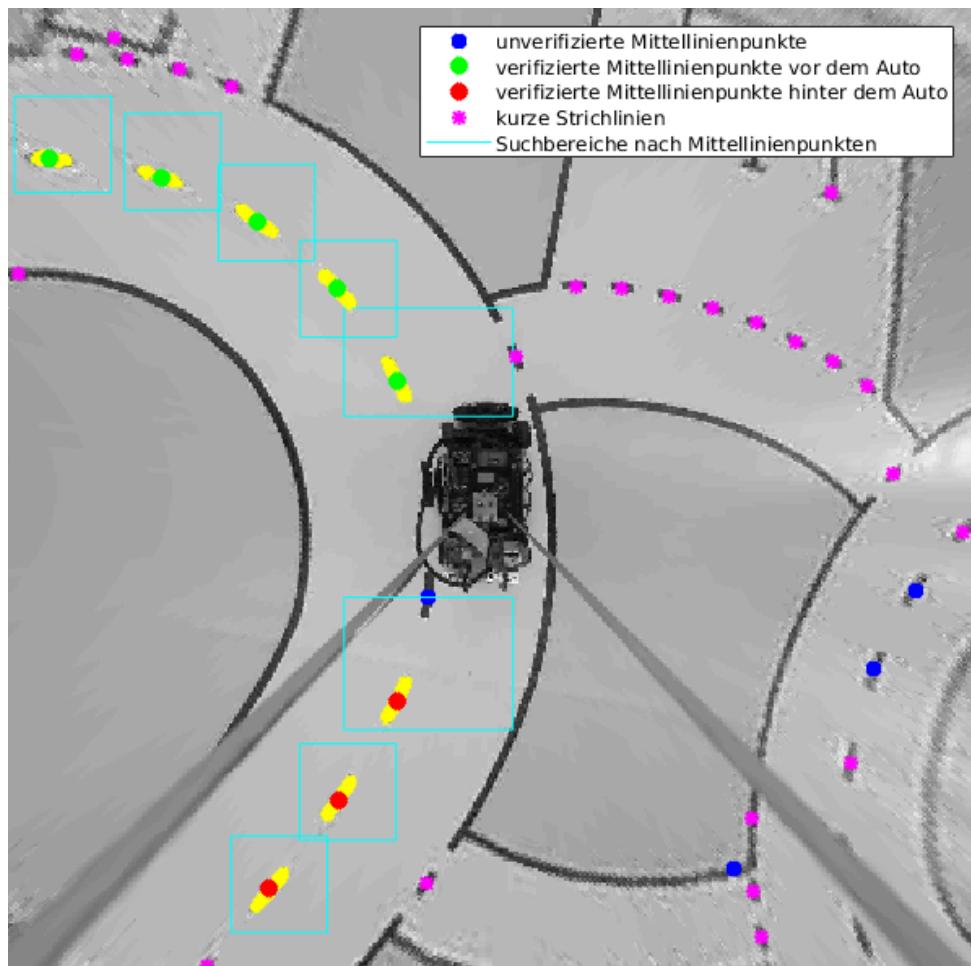


Abbildung 8.5.: Detektion von Punktgruppen mit bestimmten Eigenschaften mithilfe der *regionprops*-Funktion

### 8.3.2. seitliche Fahrbahnmarkierungen

Zur Verifikation ermittelter Randlinienpunkte werden deren Abstände zu den übrigen Fahrbahnmarkierungen untersucht und mit den im Testszenario bekannten Werten verglichen. Damit eine Randlinie als verifiziert gilt, muss ihr Abstand zu mindestens einer anderen Fahrbahnmarkierung in etwa dem Erwartungswert entsprechen.

#### 8.3.2.1. Seitenlinienvergleich M

Bei der Validierung der äußeren Fahrbahnmarkierungen aneinander wird versucht, jede gefundene Hypothese am passendsten Gegenüber zu verifizieren. Das Ergebnis der Bearbeitung einer zu prüfenden Fahrbahnmarkierung stellt den verifizierten Teil der Hypothese mit der größten validierten Punkteanzahl dar. Algorithmus 1 zeigt das Vorgehen beispielhaft für die Prüfung der linken anhand der rechten Randlinie.

#### 8.3.2.2. Beurteilung der Seitenlinien anhand der mittleren Fahrbahnmarkierung

**lineare Interpolation** Da in einem Bild signifikant weniger Mittellinienpunkte (genauer: Mittelpunkte der Mittellinienelemente) als Punkte der seitlichen Fahrbahnmarkierung gefunden werden, müssen zur Validierung der Seitenlinienpunkte durch die mittlere Fahrbahnmarkierung zusätzliche Punkte auf selbiger erzeugt werden. Hierfür wird jeweils der Mittelpunkt der Gerade zwischen zwei Mittellinienpunkten genutzt. Bei  $n$  Mittellinienpunkten entstehen also  $n - 1$  linear interpolierte Punkte.

$$\mathbf{p}_{\text{interpoliert}i} = (\mathbf{p}_{\text{real}i} + \mathbf{p}_{\text{real}i+1})/2 \quad (8.5)$$

**Algorithmus** Die Prozedur zum Vergleich der Seitenlinien mit der mittleren Fahrbahnmarkierung ist weitgehend identisch mit Abschnitt 8.3.2.1. Lediglich das Auswählen einer Hypothese der Mittellinie entfällt, da hier in jedem Fall nur ein möglicher Verlauf gefunden wird.

#### 8.3.2.3. Zusammenführung der Verifikationsmethoden

Lieferte die Überprüfung der seitlichen Fahrbahnmarkierungen aneinander die längere Koordinatenserie, so wird diese weiter verwendet. Andernfalls wird die Menge der an der mittleren Linie validierten Punkte genutzt.

**Input:** linienL(Hypothese).punkte(Punkt-Index): Punkte der linken Linie  
linienR(Hypothese).punkte(Punkt-Index): Punkte der rechten Linie  
errMax: Maximalabweichung des Linienabstands vom Sollwert  
bFsp: Fahrspurbreite

**Output:** linieVeriL(Punkt-Index): verifizierte Punkte der linken Linie

```

/* Alle Hypothesenkombinationen von linker/rechter Linie testen
 */
for i=1 to Laenge(linienL) do
    /* Laenge = Anzahl der Elemente der entsprechenden Variable
     */
    for j=1 to Laenge(linienR) do
        /* Punkte der gewählten linken Linienhypothese ihrer
           Erkennungsreihenfolge nach an gewählter rechter
           Linienhypothese verifizieren -> Abbruch bei
           Überschreitung d. maximalen Abstandsabweichung      */
        k=0
        errOK=1
        while errOK and k+1 <= Laenge(linienL(i).punkte) do
            /* Berechnung der Abstände von Punkt
               linienL(i).punkte(k+1) zu allen Punkten der rechten
               Linienhypothese linienR(j).punkte; d ist Vektor! */
            d = Abstand(linienL(i).punkte(k+1), linienR(j).punkte)
            /* Abstand Punkt - gegenüberliegende Linie ≈ Abstand
               Punkt - nächstgelegener Punkt der
               gegenüberliegenden Linie                         */
            dMin = Minimum(d)
            /* Nach Abziehen der Fahrspurbreite Betrag des Fehlers
               auswerten                                         */
            err = Betrag(dMin - 2*bFsp)
            errOK = err <= errMax
            if errOK then k=k+1;
        end
        if Laenge(linieVeriL) < k then
            /* längste verifizierbare Koordinatenserie halten      */
            linieVeriL = linienL(i).punkte(1 to k)
        end
    end
end

```

**Algorithmus 1:** Verifikation Seitenlinie-Seitenlinie

## 8.4. Sonderfall unterbrochene Randlinie M

Da das in Abschnitt 8.2 diskutierte Verfahren zur Erkennung der seitlichen Fahrbahnmarkierungen nur zum Verfolgen durchgängiger Linien konzipiert ist, einmündende Straßen jedoch durch eine unterbrochene Markierung gekennzeichnet sind, findet für diesen Sonderfall eine modifizierte Version der in Passage 8.1 sowie 8.3.1 beschriebenen Vorgehensweise zur Erkennung&Verifikation der Mittellinie Anwendung. Ist Abbruchbedingung 2 (kein Fahrbahnmarkierungspunkt auf der Scanline gefunden) des Algorithmus zur Detektion der durchgängigen Markierungen erreicht, wird ein Suchfenster (siehe pinke Quadrate in Abb. 8.4) festgelegt. Wurden in diesem Bereich durch die Matlab-Funktion *regionprops* Elemente der Länge und Breite gefunden, welche auf eine unterbrochene Randlinie hindeuten, werden diese als nächste Punkte der seitlichen Fahrbahnmarkierung vermerkt. Die Bildung des Mittelpunkts  $m$  des Suchfensters geschieht aufgrund der kurzen Länge eines Randlinienstriches nicht wie in Passage 8.3.1 auf Basis der Orientierung dessen, sondern wie in (8.2) beschrieben durch den Verschiebungsvektor zwischen den letzten zwei gefundenen Punkten der seitlichen Fahrbahnmarkierung.

Bei Fehlen eines Linienelementes im Suchfenster kann durch Bilden einer Scanline ebenfalls der Rücksprung zum Fall einer durchgängigen Randlinie vollzogen werden.

## 9. Regelung K

Dieses Kapitel beschäftigt sich mit der Bestimmung des richtigen Lenkwinkels  $\gamma$ , welcher in jeder neuen Regelungsiteration neu kalkuliert wird. Eine Iteration charakterisiert die im Kapitel 5.2.2 beschriebene Odometrie-Callback-Funktion, in welcher alle Berechnungen zur Regelung stattfinden. In jedem Aufruf wird ebenfalls die Pose des TUCar aktualisiert und der neue Lenkwinkel neben der Geschwindigkeit als Ansteuerwert zum Auto zurückgeschickt. Im Folgenden werden das zugrunde liegende Fahrzeugmodell und die Schritte zur Regelung näher beschrieben.

### 9.1. Prinzip und Fahrzeugmodell

Die Navigation des Modelfahrzeugs im Straßenverkehrsszenario basiert auf einer Weltkarte, in der alle bereits erkannten Punkte, die zu einer Fahrbahnmarkierung gehören, eingetragen wurden. Um das Auto anhand jener Punkte steuern zu können, muss dessen Standort in der Welt bekannt sein. Zu Anfang wurde im Kapitel 4.1 beschrieben, dass die Lage des Roboterkoordinatensystems  $\mathcal{B}$  im Vergleich zum Weltkoordinatensystem  $\mathcal{W}$  die Pose des Autos darstellt. Diese Pose ist der Zustandsvektor  $\mathbf{x}$  aus Position  $(x, y)$  und Orientierung  $\theta$ .

$$\mathbf{x} = \begin{pmatrix} x(t) \\ y(t) \\ \theta(t) \end{pmatrix} \quad (9.1)$$

Aus den Informationen der Weltkarte und der aktuellen räumlichen Lage des Roboters ist es möglich, eine Trajektorie zu planen, um das Fahrzeug in der Fahrspur zu halten. Daher ist es wichtig, den Zustandsvektor für jeden Regelschritt neu zu bestimmen. Zur Berechnung der neuen Position stehen uns die Geschwindigkeit  $v$  und die alte Orientierung zur Verfügung. Die Geschwindigkeit  $v$  wird intern auf dem Raspberry-PI über die Wegmessungen  $s$  des Encoders und den Zeitunterschied  $t - t_0$

in Bezug auf die vorherige Regelungsiteration ermittelt. Gemessen von der eingebauten IMU wird die Orientierung direkt in die neue Pose eingetragen. Somit ist unser Eingangsvektor  $\mathbf{u}$

$$\mathbf{u} = \begin{pmatrix} v(t) \\ \theta(t) \end{pmatrix}, \quad \text{mit } v(t) = \frac{s}{t - t_0} \quad (9.2)$$

Das TUCar ist mit Differenzialantrieb und einer Ackermann-Lenkung ausgestattet. Das Besondere an dieser Lenkung ist, dass in einer Kurvenfahrt das innen liegende Vorderrad stärker einlenkt als das Äußere, sodass sich ein gemeinsamer Schnittpunkt mit allen vier Radachsen ergibt (siehe Abb. 9.1). Dieser Punkt wird auch als Instantaneous Centre of Rotation (ICR) bezeichnet und beschreibt den Mittelpunkt der gefahrenen Kreisbahn.

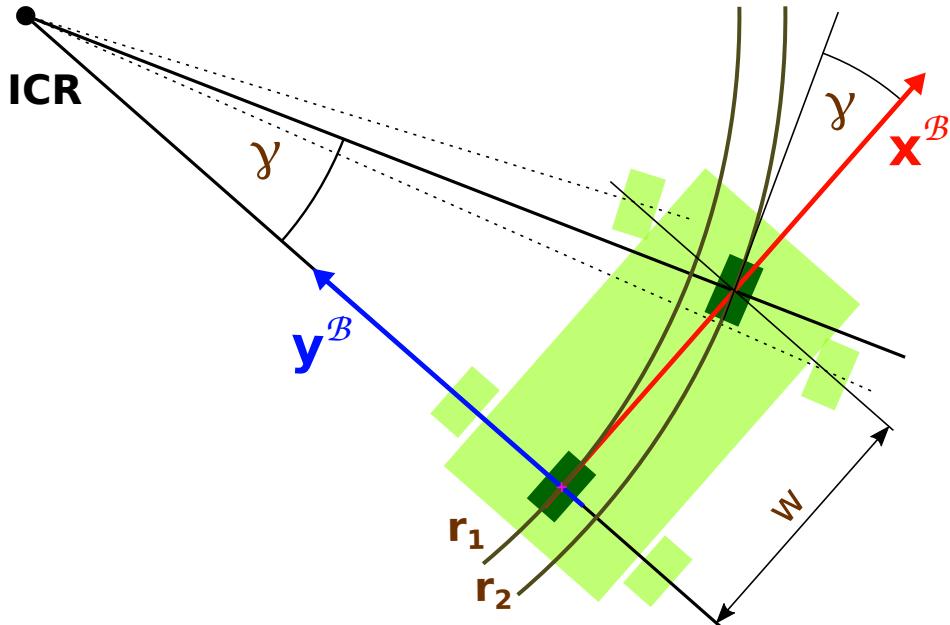


Abbildung 9.1.: Modellierung der Ackermannlenkung durch übliche Vereinfachung auf das Bicycle-Modell in Anlehnung an (Corke 2017)

Zur leichteren mathematischen Beschreibung des Fahrzeugs wird das Modell üblicherweise auf das eines Zweirades reduziert (*Bicycle-Modell*). Der Kreisradius ist dann der Abstand vom ICR zum Ursprung des Roboterkoordinatensystems. Außerdem lässt sich  $\gamma$  wie in Abbildung 9.1 gezeigt im Winkel der beiden Radachsen zueinander wieder-

finden. Nachfolgende Formel (9.3) zum vereinfachten Fahrzeugmodell in Weltkoordinaten  $\mathcal{W}$  ist aus (Corke 2017) entnommen. Der Eingangsvektor besteht neben der Geschwindigkeit  $v(t)$  allerdings nicht aus dem Lenkwinkel  $\gamma$ , sondern aus der Orientierung  $\theta$ , weshalb bei uns die Berechnung von  $\theta$  aus  $\gamma$  nicht zum Einsatz kommt.

$$\dot{\mathbf{x}} = \begin{pmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\theta}(t) \end{pmatrix} = \begin{pmatrix} v \cdot \cos \theta \\ v \cdot \sin \theta \\ \frac{v}{w} \cdot \tan \gamma \end{pmatrix} \quad (9.3)$$

Das Modelfahrzeug wird nach dem „Pure-Pursuit“-Regler gesteuert. In einem bestimmten Abstand vor dem Auto wird ein Zielpunkt gesetzt, zu welchem bis zur Erstellung eines neuen Zielpunktes navigiert werden soll. Da wir den Schlupf der Räder vernachlässigt haben, ist das Erreichen des Zielpunktes völlig unabhängig von der Geschwindigkeit und nur noch an den Lenkwinkel gebunden. Dieser wird so eingestellt, dass die dadurch gefahrene Kreisbahn durch den Zielpunkt verläuft. Wie der Zielpunkt und der dazu passende Lenkwinkel  $\gamma$  ermittelt werden, beschreiben die nun folgenden zwei Abschnitte.

## 9.2. Zielpunktgewinnung M

Um den Zielpunkt für den implementierten „Pure-Pursuit“-Regler zu erhalten, müssen die nachfolgend beschriebenen Schritte ausgeführt werden. Abschnitt 9.2.1 beleuchtet Kriterien, nach denen aus allen in der Weltkarte vorhandenen Punkten für die Regelung notwendige selektiert werden. Passage 9.2.2 beschäftigt sich mit der Vorgehensweise, die gewählten Punkte einer Fahrbahnmarkierung in ein kompakt parametrierbares mathematisches Modell (Kreissegment) zu überführen, um diese weiter verarbeiten zu können. Die Kapitel 9.2.3 sowie 9.2.4 klären die Frage, wie diese drei Kreissegmente zu einer Trajektorie zusammengeführt werden. Final wird in Passus 9.2.5 erläutert, wie aus dieser der Zielpunkt extrahiert wird.

### 9.2.1. Holen der benötigten Punkte aus der Weltkarte

Im gewählten Fahrspurverfolgungskonzept läuft die Erkennung der Fahrbahnmarkierungen und die Regelung des Fahrzeugs asynchron ab, beide Komponenten kommunizieren nur über eine Weltkarte miteinander. Somit muss vor jedem Regelzyklus

festgestellt werden, welche Punkte dieser Weltkarte zum Steuern des Roboters noch relevant sind. Hierfür wurden einige Regeln festgelegt:

1. Der Zeitpunkt, an welchem die Punkte in die Weltkarte eingetragen wurden, muss aktuell genug sein.
2. Der Abstand der aktuellen Roboterpose zur Pose, an welcher die Punkte in die globale Karte eingetragen wurden, darf einen Grenzwert nicht überschreiten.
3. Die x-Koordinate der in das Roboter-KS  $\mathcal{B}$  transformierten Punkte muss in einem bestimmten Intervall liegen.

Um die Punkte nach Kategorie 3 filtern zu können und später zur Regelung zu nutzen, werden sie nach Schritt 2 vom Welt-KS  $\mathcal{W}$  in Roboterkoordinaten (KS  $\mathcal{B}$ ) transformiert.

### 9.2.2. Kreissegment-Fit

Um später alle drei Fahrbahnmarkierungen zur Zielpunktgewinnung in die gewünschte Fahrspur verschieben zu können, muss eine mathematische Repräsentation der Mittellinie und Seitenlinien gefunden werden, welche dies erlaubt. Da in der Weltkarte aus mehreren Bildern erkannte Punkte vorhanden sind, soll ein zu allen diesen Frames möglichst gut passender Zielpunkt gefunden werden. Wählt man den Bereich der für die Zielpunktfindung relevanten Fahrbahnmarkierungen nicht zu groß, kann deren Verlauf gut durch Kreisbögen approximiert werden.

#### 9.2.2.1. Kreisfit

Zur Ermittlung des optimalen Kreises durch die gefragten Punkte wurde der Taubin-Kreisfit (Taubin 1991) in der Implementierung von (Nikolai Chernov 2012) verwendet.

#### 9.2.2.2. Ermittlung zusätzlicher Parameter

Neben dem Mittelpunkt  $\mathbf{m}$  und Radius  $r$  muss zur vollständigen Parameterisierung des Kreisbogens Start- und Endwinkel sowie die Drehrichtung des Bogens bekannt sein.

**Start- und Endwinkel** Zur Ermittlung des Startwinkels  $\varphi_s$  wird vom Mittelpunkt des Kreises  $m$  zum Punkt  $p_s$  ein Vektor  $o_s$  aufgespannt.  $p_s$  stellt den Punkt mit dem geringsten Abstand vom Koordinatenursprung des Roboter-KS  $\mathcal{B}$  aus der Koordinatenserie, welche zum Kreisfit genutzt wurde, dar (s. Abb. 9.2). Der gesuchte Startwinkel ergibt sich als:

$$\varphi_s = \text{atan}2(o_{sy}, o_{sx}) = \text{atan}2(p_{sy} - m_y, p_{sx} - m_x) \quad (9.4)$$

Zur Ermittlung des Endwinkels  $\varphi_e$  wird analog unter Nutzung des Punktes mit dem größten Abstand vom Koordinatenursprung des Roboter-KS  $\mathcal{B}$  verfahren.

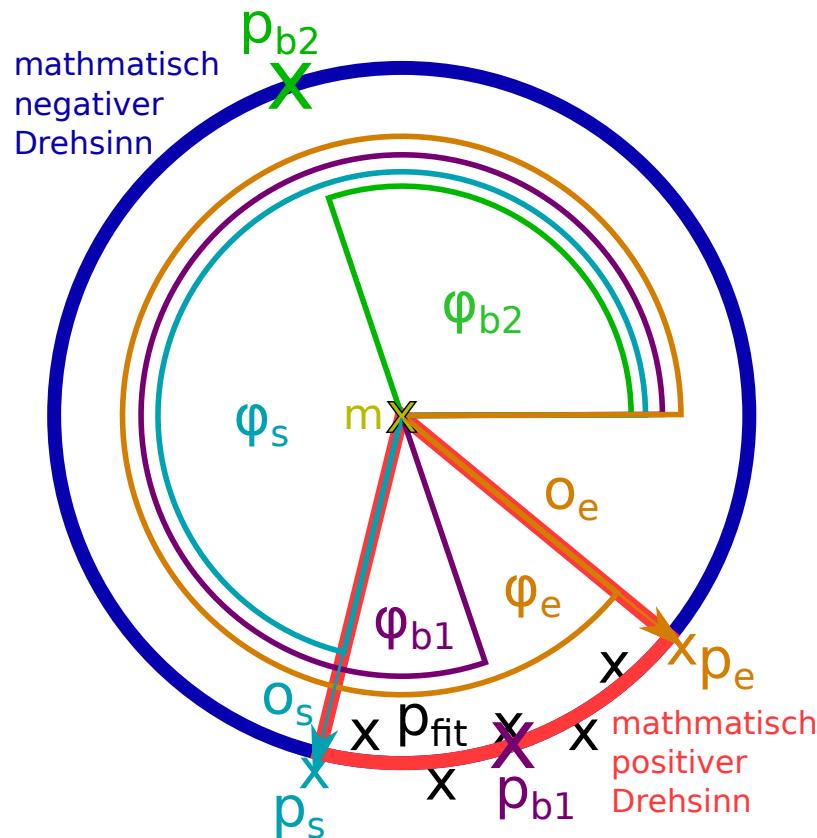


Abbildung 9.2.: Ermittlung der Rotationsrichtung eines Kreissegments

**Rotationsrichtung** Die Methode zur Ermittlung der Rotationsrichtung orientiert sich an (Drauschke 2016). Es wird der Punkt  $p_{b1}$  (9.5) auf dem Radius des Kreises

mit dem Winkel  $\varphi_{b1} = (\varphi_s + \varphi_e)/2$  errechnet. Desweiteren wird der Punkt  $\mathbf{p}_{b2}$  (9.5) auf dem Radius des Kreises mit dem Winkel  $\varphi_{b2} = (\varphi_s + \varphi_e)/2 + \pi$  bestimmt (s. Abb. 9.2).

$$\mathbf{p}_{b1/2} = \mathbf{m} + \begin{pmatrix} \cos \varphi_{b1/2} \\ \sin \varphi_{b1/2} \end{pmatrix} \cdot r \quad (9.5)$$

Aus den Mittelwerten  $\|\bar{\mathbf{a}}\|_{1/2}$  (9.6) der Abstände  $\|\mathbf{a}\|_{1/2i} = \|\mathbf{p}_{b1/2} - \mathbf{p}_{fiti}\|$  der  $n$  zum Kreisfit genutzten Punkte  $\mathbf{p}_{fiti}$  zu den Punkten  $\mathbf{p}_{b1}$  und  $\mathbf{p}_{b2}$  kann nun die Rotationsrichtung bestimmt werden.

$$\|\bar{\mathbf{a}}\|_{1/2} = \frac{\sum_{i=1}^n \|\mathbf{a}\|_{1/2i}}{n} \quad (9.6)$$

$\ \bar{\mathbf{a}}\ _1 < \ \bar{\mathbf{a}}\ _2$	$\ \bar{\mathbf{a}}\ _1 \geq \ \bar{\mathbf{a}}\ _2$
$\varphi_s < \varphi_e$	mathematisch positiver Drehsinn
$\varphi_s \geq \varphi_e$	mathematisch negativer Drehsinn

### 9.2.3. Verschiebung der Kreissegmente

Um die drei gefundenen Kreissegmente (in Abb. 9.3 als blaue, rote und grüne Strickpunktlinien inklusive umliegender x derselben Farbe zur Verkörperung der zum Fit genutzten Punkte dargestellt) zur Gewinnung eines Zielpunktes nutzen zu können, müssen diese in die Mitte der gewünschten Fahrbahn (links/rechts) verschoben werden. Hierfür muss je nach Rotationsrichtung des Kreissegments lediglich der Radius dessen um die halbe bzw. anderthalbe Fahrspurbreite modifiziert werden, es entstehen die sich schneidenden/übereinanderliegenden roten, blauen und grünen Kreissegmente in Abbildung 9.3. Zur genauen Aufschlüsselung der Logik zur Radiusänderung sei der Leser auf den Quellcode verwiesen. REFERENZ

### 9.2.4. Bildung des Trajektorienkreissegments

Um aus den drei in die Fahrspur verschobenen Segmenten einen Kreisbogen zu erhalten, werden die in die Fahrspur verschobenen Kreissegmente nun gesampelt. Von jedem Kreisegment werden so viele Punkte abgetastet, wie zur Berechnung seiner Parameter genutzt wurden. Die so entstandene Koordinatenreihe bildet die Basis für den Fit des Trajektoriekreisbogens (in Abb. 9.3 als lila Punktlinie dargestellt). Hierfür wird dieselbe Prozedur wie in Abschnitt 9.2.2 verwendet.

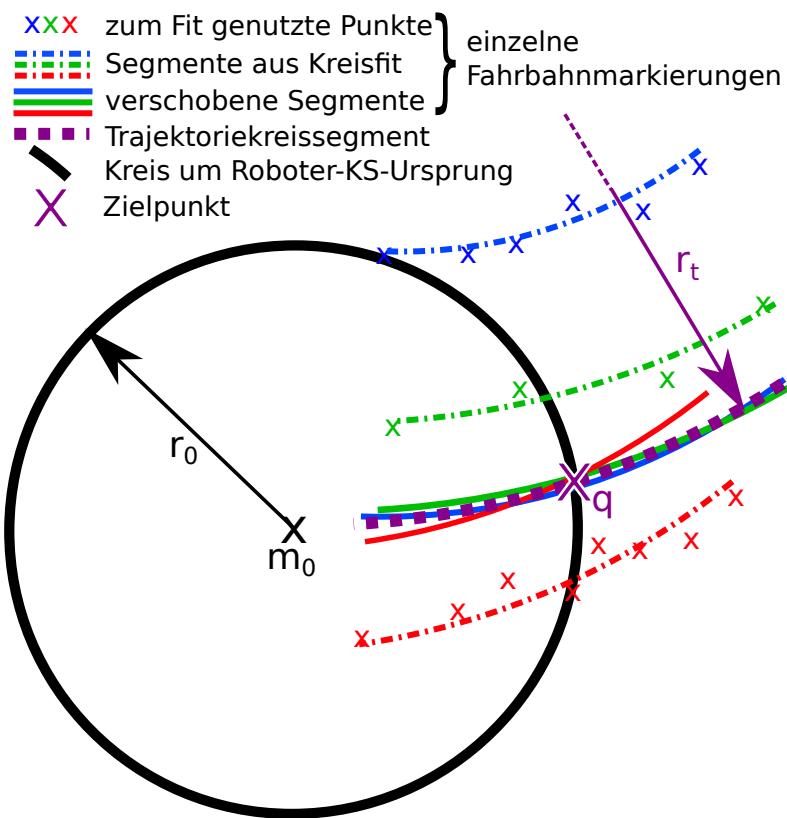


Abbildung 9.3.: Vorgehensweise der Zielpunktgewinnung

Kreissegment-Fit über Punkte der einzelnen Fahrbahnmarkierungen  
 $\Rightarrow$  Verschiebung der Kreissegmente in Fahrspur  $\Rightarrow$  Sampeln der Kreissegmente in Fahrspur  $\Rightarrow$  Fit des Trajektoriekreissegments  $\Rightarrow$  Schnittpunktberechnung mit Kreis um Roboter-KS-Ursprung

### 9.2.5. Schnittpunktberechnung

Um eine möglichst gleichmäßige Regelung des Lenkwinkels zu ermöglichen, sollte der Zielpunkt  $\mathbf{q}$  der „Pure-Pursuit“-Regelung in jeder Fahrsituation den gleichen Abstand vom Ursprung des Roboter-KS  $\mathcal{B}$  besitzen (die sogenannte Lookahead-Distance).  $\mathbf{q}$  muss sich also auf dem Kreis mit Mittelpunkt  $\mathbf{m}_0 = (0, 0)$  und Radius  $r_0$  gleich der Lookahead-Distance befinden (schwarz in Abb. 9.3 dargestellt). Da  $\mathbf{q}$  ebenfalls auf dem Trajektorienkreissegment mit Mittelpunkt  $\mathbf{m}_t$  und Radius  $r_t$  (lila in Abb. 9.3 verzeichnet) liegt, reduziert sich die Zielpunktermittlung auf die Schnittpunktberechnung zweier Kreise (Wikipedia 2018b):

$$\mathbf{q}_{1/2} = \mathbf{m}_0 + \mathbf{d}_1 \pm l \cdot \mathbf{e}_2 \quad (9.7a)$$

$$\mathbf{e}_2 = \begin{pmatrix} -e_{1y} \\ e_{1x} \end{pmatrix} \quad (9.7b)$$

$$\mathbf{e}_1 = \begin{pmatrix} e_{1x} \\ e_{1y} \end{pmatrix} = \frac{\mathbf{m}_t - \mathbf{m}_0}{\|\mathbf{m}_t - \mathbf{m}_0\|} \quad (9.7c)$$

$$\mathbf{d}_1 = \begin{pmatrix} d_{1x} \\ d_{1y} \end{pmatrix} = \frac{1}{2} \cdot \left( \frac{r_t - r_0}{\|\mathbf{m}_t - \mathbf{m}_0\|^2} + 1 \right) \cdot (\mathbf{m}_t - \mathbf{m}_0) \quad (9.7d)$$

$$l = \sqrt[2]{r_0^2 - \|\mathbf{d}_0\|^2} = \sqrt[2]{r_t^2 - \|\mathbf{d}_t\|^2} \quad (9.7e)$$

Da zwei sich schneidende Kreise zwei Schnittpunkte besitzen, muss nun geprüft werden, welcher der Schnittpunkte auf dem gültigen Segment des Trajektorienkreises liegt. Dies kann anhand von Start- und Endwinkel sowie dessen Rotationsrichtung ermittelt werden und ist im Quelltext REFERENZ genauer nachzulesen.

## 9.3. Berechnung des Lenkwinkels K

Bekannt sind momentan der anzusteuernde Zielpunkt, die Pose des Roboters sowie dessen Radstand  $w$ . Anschließend soll der Zusammenhang zwischen Lenkwinkel und Zielpunkt hergeleitet werden. Aus der in Abb. 9.1 dargestellten Geometrie lässt sich folgender Zusammenhang ablesen:

$$\tan \gamma = \frac{w}{r_1} \quad (9.8)$$

Um einen Kreis eindeutig definieren zu können, benötigt man mindestens drei bekannte Parameter. Der noch unbekannte Radius  $r_1 = r$  (siehe Abb. 9.1) lässt sich aus dem Zielpunkt, der Position und der Orientierung des Autos bestimmen. Abbildung 9.4 zeigt eine anschauliche Möglichkeit zur Herleitung der gesuchten Beziehung.

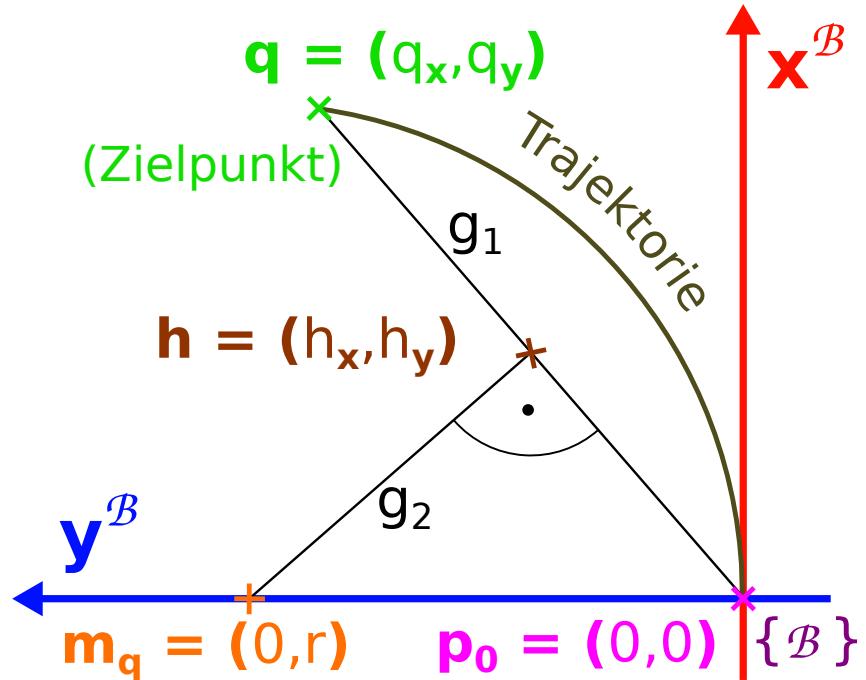


Abbildung 9.4.: Visualisierung zur Bestimmung des Kreisradius aus dem gegebenen Zielpunkt  $q$  in Roboterkoordinaten

Es lassen sich zwei Hilfsgeraden  $g_1$  und  $g_2$  aufstellen.  $g_1$  verläuft durch die Position des Roboters  $p_0$  sowie den Zielpunkt  $q$  und stellt somit die Sehne des Kreisbogens der geplanten Trajektorie dar. Die Mittelsenkrechte dieser Sehne bildet die Gerade  $g_2$ .

$$\text{Gerade } g_1 : y = m_q \cdot x \quad (9.9a)$$

$$\text{Gerade } g_2 : y = m_h \cdot x + n_h \quad (9.9b)$$

Für den Schnittpunkt  $\mathbf{h}$  gilt

$$h_x = 0.5 \cdot \mathbf{q}_x \quad (9.10a)$$

$$h_y = 0.5 \cdot \mathbf{q}_y \quad (9.10b)$$

und für die Steigungen der Geraden gelten

$$m_{\mathbf{q}} = \frac{\Delta y}{\Delta x} = \frac{\mathbf{q}_y}{\mathbf{q}_x} \quad (9.11a)$$

$$m_h = \frac{\Delta y}{\Delta x} = \frac{\mathbf{q}_x}{-\mathbf{q}_y} \quad (9.11b)$$

Die  $y^B$ -Achse und die Gerade  $g_2$  stehen jeweils senkrecht auf dem Kreisbogen und müssen sich demzufolge im Kreismittelpunkt  $\mathbf{m}_q$  schneiden. Da  $\mathbf{m}_q$  auf der  $y^B$ -Achse liegt ist die  $x^B$ -Koordinate mit „0“ bereits bekannt. Damit steht der gesuchte Radius  $r$  in der  $y^B$ -Komponente des Mittelpunktes, was wiederum bedeutet, dass auch  $r = n_h$  gilt. Setzt man mit diesem Wissen den Punkt  $\mathbf{h}$  in die Geradengleichung von  $g_2$  (9.9b) ein, erhält man durch Umstellen die Formel (9.12) für den Radius  $r$ .

$$r = n_h = h_y - m_h \cdot h_x = 0,5 \cdot \left( q_y + \frac{q_x^2}{q_y} \right) \quad (9.12)$$

Ausgehend von Gleichung (9.8) lässt sich nun der gesuchte Lenkwinkel mit Hilfe der Arcustangensfunktion bestimmen.

$$\gamma = \arctan \left( \frac{w}{r} \right) \quad (9.13)$$

## 9.4. Implementierung M

Da die theoretischen Grundlagen zum Aufbau des „Pure-Pursuit“-Reglers eines Fahrzeugs mit Ackermannlenkung auf Basis einer Weltkarte nun bekannt sind, soll nun die Eingliederung dieser in das vorliegende System erläutert werden. Wie in Kapitel 5.2.2 bereits beschrieben, erfolgt die Regelung im Rahmen des sogenannten

---

Odometrie-Callbacks. Diese Funktion wird gerufen, sobald der ROS-Node des Roboters neue Daten im entsprechenden Odometrie-Topic veröffentlicht, die (einstellbare) Frequenz des Publizierens dieser Informationen bestimmt also die Regelfrequenz.

Neben der in der empfangenen Message selbst enthaltenen Geschwindigkeit wird für die Aktualisierung der im Folgenden benötigten Pose (siehe Abschnitt 9.1) noch die Orientierung des Fahrzeugs aus dem IMU-Topic abgerufen. Auf Basis dieser Position & Orientierung können nun wie in den Abschnitten 9.2 und 9.3 erläutert regelungsrelevante Punkte aus der Weltkarte extrahiert und auf Basis dieser Zielpunkt und Lenkwinkel ermittelt werden.

**Vorteile der Architektur** Die durch Kopplung der Fahrspurerkennung und Regelung an unterschiedliche Ereignisse erreichte Trennung dieser beiden Prozesse macht es möglich, die Regelung des Roboters mit geringeren Totzeiten und höheren Frequenzen als bei einer Fusion dieser Algorithmen in einer Funktion durchzuführen. Fahrbahnmarkierungen müssen vor Ausführen der Regelung nicht erst erkannt werden, stattdessen wird die Lage dieser bereits in der Weltkarte eingetragenen Koordinatenserien in Bezug auf das Modelfahrzeug bestimmt. Wurden die Fahrbahnmarkierungen richtig erkannt und ist das ermittelte Bewegungsmodell zur Posenaktualisierung ausreichend genau, so ergibt sich trotz relativ niedriger Bildrate kaum ein Nachteil zu einer der Regelfrequenz gleichen Häufigkeit des Verarbeitens neuer Aufnahmen. Ein kritischer Punkt wird erst erreicht, wenn die gefahrene Entfernung zwischen zwei Bildern einen Zielpunkt außerhalb des Bereichs der schon erkannten Fahrbahnmarkierungen erfordert.

## 10. Evaluation K

In diesem Kapitel wird das final zur Fahrspurverfolgung genutzte Vorgehen ausführlich bewertet. Zu Beginn wird auf prinzipielle Vor- und Nachteile eingegangen, um schließlich die Performance anhand einiger Tests zu demonstrieren. Neben einer zeitlichen Betrachtung wird die Robustheit des Riverflow-Algorithmus bezüglich Geschwindigkeit, Störobjekten und abgedeckten Bereichen in der Fahrspur getestet.

Abschnitt 10.2 beleuchtet die Schwierigkeiten, ein entwickeltes System ohne Vergleichswerte oder Messsysteme objektiv zu bewerten. Anschließend sollen charakteristische Messwerte der Fahrspurerkennung, Regelung, sowie dem Zusammenspiel beider Komponenten dargestellt und diskutiert werden. In Paragraph 10.3 wird die Laufzeit der Fahrspurverfolgungskomponenten ermittelt und auf Basis dieser Messwerte die maximal verarbeitbare Bildfrequenz eingestellt. Passus 10.4 vergleicht die erzeugte Weltkarte mit der Zeichnung des Testszenarios um Rückschlüsse auf die Qualität der am Kartenaufbau beteiligten Messungen zu ziehen. In Passage 10.5 soll das Verhalten des Modellautos bei unterschiedlichen Geschwindigkeiten evaluiert werden.

Die anknüpfend folgenden Abschnitte stellen eine größtenteils qualitative Evaluation des Fahrverhaltens bei Veränderung des Testszenarios dar. Da fortan das Straßenszenario vielseitiger genutzt werden soll, ist es denkbar, dass sich künftig zusätzliche Objekte im Sichtbereich des Roboters befinden. Daher untersucht Absatz 10.6 die Reaktion der Fahrspurverfolgung auf das Hinzufügen von Objekten zur Modellwelt, während in Passage 10.7 die Robustheit bei Reduktion der Kennzeichen einer Straße betrachtet wird.

**Überlegungen zur Darstellung der Messwerte M** Um möglichst viele Informationen kompakt abzubilden, wurden unter anderem die Diagrammtypen Boxplot sowie Histogramm genutzt. Da kurz nach dem Start des Fahrzeugs

- MATLAB viele Funktionen erst noch in den Hauptspeicher laden muss („Warmlaufphase“)

- 
- Das Fahrzeug vor vollendeter Verarbeitung des ersten Bildes schon ohne gültigen Zielpunkt losfährt

wurden diese nicht-repräsentativen Phasen in den entsprechenden Grafiken ausgelassen, da sie die Aussagekraft der Messwerte beeinträchtigen würden. In der Bildunterschrift wurde jeweils das Stichwort „ohne Startphase“ eingefügt.

### 10.1. Diskussion prinzipieller Gesichtspunkte M

Die Fahrbahnmarkierungserkennung mittels Riverflow-Algorithmus besitzt drei signifikante Unterschiede zu den vorhergehenden Konzepten:

1. **Linien-Repräsentation** Dem Straßenverlauf wird kein mit wenigen Parametern beschreibbares, polynomiales Modell zugrunde gelegt. Stattdessen wird auf eine diskrete Abbildung durch entsprechende Stützstellen zurückgegriffen, die bestimmte Bedingungen erfüllt:
  - kleine Orientierungsänderung der Verbindungsvektoren aufeinanderfolgender Punkte
  - ähnlicher Abstand konsekutiver Stützstellen
2. **Notwendige Vorinformationen** Durch die Verwendung von Algorithmen, die die Lage der Fahrbahnmarkierungen in beliebigen Bildern ohne zeitintensive/fehleranfällige Initialisierungsroutinen bestimmen können, sind keine Informationen aus vorherigen Aufnahmen zur Linienfindung notwendig. Hierbei wird besonders die Kenntnis der Position des Roboters in der Fahrspur und Modellwissen zu speziellen Eigenschaften der Straße genutzt (s. Anfang Kapitel 8).
3. **Verifikation** Nach der initialen Erkennung wird eine Plausibilitätsprüfung der erkannten Linienverläufe vorgenommen. Der Abstand eines Punktes zu mindestens einer weiteren Fahrbahnmarkierung muss der bekannten, im Testszenario vorgegebenen Distanz gleichen. Ein ähnliches Konzept hätte auch bei einem polynomisierten Verfahren welches drei Einzellinien erkennt (z.B. der Ransac-Ansatz aus Kapitel 7.1) verwirklicht werden können. Die Implementierung wurde jedoch erst durch die Möglichkeit der Erkennung mehrerer po-

tenzieller Verläufe der seitlichen Fahrbahnmarkierungen durch den Riverflow-Algorithmus notwendig, da hier eine Auswahl der besten Hypothese erfolgen muss.

Die beschriebenen fundamentalen Differenzen führen zu folgenden Vor- und Nachteilen:

**Vorteile** Punkt 1 (Linien-Repräsentation) macht es möglich, nahezu jeden Straßenverlauf ausreichend präzise zu approximieren:

- Kurven mit über  $90^\circ$  Richtungsänderung können nun erkannt werden. Dies führte mit dem Polynom  $y(x)$  der vorhergehenden Herangehensweisen zu Problemen, da  $90^\circ$ -Kurven einen unendlichen Anstieg der gesuchten kubischen Funktion zur Folge hätten.
- Da die Strecke in guter Näherung aus Kreissegmenten und Geraden aufgebaut ist, kann das Polynom  $y(x)$  der ersten beiden Konzepte diesen Verlauf nur annähern. Die nun untersuchte Vorgehensweise hingegen findet Punkte, die exakt mittig auf der Fahrbahnmarkierung liegen.

Unterschied 2 (Notwendige Vorinformationen) verhindert, dass eine Fehlerkennung unmittelbare Auswirkungen auf die Detektion von Linienverläufen im nächsten Bild hat. Ein „Weglaufen“ der Masken oder des Systemzustands  $x$  wird unterbunden.

Punkt 3 (Verifikation) bietet zusätzliche Sicherheit, da nur in sich stimmige Straßenverläufe in die Weltkarte eingetragen werden. Außerdem wird bei Erkennung mehrerer Hypothesen für die seitlichen Fahrbahnmarkierungen die Auswahl der besten erkannten Koordinatenserie möglich.

**Nachteile** Der in Unterschied 1 (Linien-Repräsentation) beschriebene Verzicht auf ein kompaktes mathematisches Fahrspurmodell führt zum Abbruch des Algorithmus beim zu schwachen Auftreten eines Stücks der gesuchten Linie im gefilterten Bild. Die genutzte iterative Methode, welche auf Basis der schon gefundenen Punkte die mögliche Lage der weiteren Fahrbahnmarkierung eingrenzt, geht von einem Ende der gesuchten Linie aus, sobald kein Punkt auf der nächsten Scanline gefunden wurde (siehe Punkt 2 in Abschnitt 8.2.4). Durch die gute Qualität der Aufnahmen im Testszenario ohne eingebrachte Störeinflüsse stellt dies jedoch kein Problem dar.

Punkt 2 (Notwendige Vorinformationen) bietet Platz für Optimierungsmöglichkeiten des Algorithmus. Es wäre möglich, in jedem Bild die Verarbeitung auf bisher unbekannte Bereiche der Aufnahme zu reduzieren. Startpunkte für die Algorithmen könnten aus bereits in die Weltkarte eingetragenen Koordinaten ermittelt werden. Der erworbene Geschwindigkeitszuwachs eliminiert jedoch die Redundanz gefunderner Punkte, was die Genauigkeit und Zuverlässigkeit der momentan implementierten Vorgehensweise vermindert.

Unterschied 3 (Verifikation) und die dadurch ermöglichte Verfolgung mehrerer Hypothesen der seitlichen Fahrbahnmarkierungen können die Dauer dieses Verarbeitungsschrittes vervielfachen (siehe Abb. 10.14).

## 10.2. Evaluation ohne Ground Truth

Für die Detektion von Fahrspuren auf Einzelbildern und die daraus resultierende Bewegung des Fahrzeugs existieren zum jetzigen Zeitpunkt keine Vergleichswerte.

**Bildverarbeitung** Für ein Benchmark der Fahrspurerkennung könnte ein Ground-Truth-Datensatz in Form von Bildern mit richtig eingetragenen Fahrbahnmarkierungen die präziseste Bewertung hervorbringen. Da die Erstellung eines solchen jedoch sehr zeitaufwendig ist, wurden größtenteils qualitative Methoden zur Evaluation gewählt.

**Regelung** Die Güte der Regelung bzw. des Zusammenspiels aus Bildverarbeitung und Regelung könnte quantitativ exakt durch ein hinreichend genaues System zur Lokalisierung des Fahrzeugs im Testszenario ermittelt werden. Somit wäre es möglich die Abweichung der gefahrenen Strecke von einer vorgegebenen Ideallinie auszuwerten. Da die Inbetriebnahme des an der Professur vorhandenen zweidimensionalen Tracking-Systems viel Einarbeitungszeit benötigt, wird hier auf eine einfache Methode zurückgegriffen. Es wird vermerkt ob ein gültiger Zielpunkt für die Regelung ermittelt werden konnte (s. Abschnitt 9.2), sowie ob das Fahrzeug die Fahrspur völlig verlässt oder dem Straßenverlauf zufriedenstellend folgt.

## 10.3. Laufzeit M

Dieser Paragraph beschäftigt sich mit der Ermittlung der maximal verarbeitbaren Bildfrequenz. Durch die Auswertung der Laufzeit der Bildverarbeitungs- und Regelungskomponente soll die Aufrufhäufigkeit dieser Funktionen so eingestellt werden, dass durch volle Ausnutzung der zur Verfügung stehenden Rechenzeit die schnellstmögliche sichere Fahrt im Testszenario möglich wird. Außerdem soll der Bestandteil der Bildverarbeitungsfunktion mit dem größten Zeitbedarf identifiziert und, wenn möglich, durch Parameteranpassung oder minimale Quellcodeeingriffe optimiert werden.

Den Ausgangspunkt für die folgenden Optimierungen bildet eine Geschwindigkeit von  $0.1 \text{ m s}^{-1}$ . Die Frequenz  $f_{img}$ , mit welcher die Bilder verarbeitet werden, wurde auf 1 Hz festgelegt, da die Extraktion der Linienpunkte eines Testbildes ca. 0.5 s benötigt. Die Frequenz der Regelung  $f_{odom}$  wurde initial auf 20 Hz festgelegt, da der Rechenaufwand für einen Regelungszyklus sehr gering ist. Eine noch höhere Regelfrequenz verspricht keinen Performancegewinn und läge schon nahe der maximalen Ansteuerfrequenz des Lenkservos sowie der Abtastrate der IMU (jeweils 50 Hz).

Mithilfe dieser Einstellung absolvierte das Fahrzeug erfolgreich eine Runde im Testparcours. Wie in Abb. 10.1d zu sehen, benötigt ein Bild-Callback im Echtzeitbetrieb, d.h. bei mehrmaliger Ausführung erheblich weniger Zeit als bei einmaliger Messung anhand eines Testbildes. Die Verringerung der Bearbeitungsdauer  $t_{img}$  von ca. 500 ms auf 128 ms (Median der in Abb. 10.1d dargestellten Daten) lässt sich unter anderem durch die Fähigkeit MATLABs, Funktionen bei wiederholter Ausführung im Hauptspeicher zu halten und somit schneller ausführen zu können, erklären. Eine signifikante Erhöhung der Bildfrequenz  $f_{img}$  ist also möglich. Nimmt man die von der Regelung und Posenaktualisierung, d.h. dem Odometrie-Callback benötigte Zeit als konstant an, lässt sich die theoretisch mögliche Bildfrequenz wie folgt berechnen:

$$T_{img} = t_{img} + t_{odom-per-sec} \cdot T_{img} \quad (10.1a)$$

$$T_{img} = \frac{t_{img}}{1 - t_{odom-per-sec}} \quad (10.1b)$$

$$f_{img} = \frac{1}{T_{img}} \quad (10.1c)$$

Die vom Odometrie-Callback pro Sekunde in Anspruch genommene Zeit  $t_{odom-per-sec}$

wurde direkt als Median der in Abb. 10.1d dargestellten Messwerte errechnet, könnte aber bei einer von 1 Hz abweichenden initialen Bildfrequenz auch auf Basis der Dauer eines Odometrie-Callbacks  $t_{odom}$  bestimmt werden:

$$t_{odom-per-sec} = t_{odom} \cdot f_{odom} \quad (10.2)$$

Mit einer Bildbearbeitungszeit von  $t_{img} = 128$  ms sowie dem Zeitbedarf des Odometrie-Callbacks von  $t_{odom-per-sec} = 262$  ms s<sup>-1</sup>, ergibt sich eine in der Theorie nutzbare Framerate von über 5 Hz. Mit dieser Bildfrequenz wurde eine weitere Testfahrt absolviert, in Abb. 10.1a wird die bessere Nutzung der zur Verfügung stehenden Rechenzeit deutlich. Abbildung 10.1a zeigt jedoch auch, dass die Periodendauer der Bildverarbeitung zwischen 2 Punkten oszilliert. Der erwartete zeitliche Abstand zweier konsekutiver Aufnahmen von 200 ms wird häufig nicht erreicht, stattdessen befindet sich ein Abstand von  $2 \cdot 200$  ms = 400 ms zwischen aufeinander folgenden Bildern. Dies führt zu dem Schluss, dass durch nicht vorhandene, da kontraproduktive Pufferung Aufnahmen ausgelassen wurden.

Im Folgenden wurde die Bildfrequenz zuerst auf 4 (s. Abb. 10.1b), später auf 3 Hz verringert (s. Abb. 10.1c). Erst jetzt konnte kein Springen der Periodendauer des Bild-Callbacks auf ein Vielfaches seines Erwartungswertes mehr festgestellt werden. Eine leichte Oszillation des zeitlichen Abstands zweier Aufnahmen ist auch in Abb. 10.1c zu erkennen, jedoch beträgt der Mittelwert fast exakt die erwarteten 333 ms. Die verbleibenden Unregelmäßigkeiten sind auf den Kamera-Node, welcher die Bilder aufnimmt und im entsprechenden Topic veröffentlicht, zurückzuführen (s. Abb. 10.2 und Abschnitt 5.1.1). Abbildung 10.3 stellt zusammenfassend die pro Bild mit Regelung, Bildverarbeitung, sonstigem Overhead & im Leerlauf verbrachte Zeit einer gefahrenen Runde dar.

**Dauer einzelner Komponenten der Fahrspurerkennung** Um in Zukunft Optimierungen an den implementierten Bildverarbeitungsalgorithmen durchführen zu können, ist es unumgänglich, nicht nur die gesamt benötigte Zeit, sondern auch die Dauer der einzelnen Module zu messen.

Diese Untersuchungen führten zu interessanten Ergebnissen, welche die Echtzeitfähigkeit der MATLAB-ROS-Schnittstelle in Frage stellen. Wie in Abbildung 10.4c zu sehen, wird ein Großteil der Dauer eines Bild-Callbacks zum Umwandeln der ROS-Message

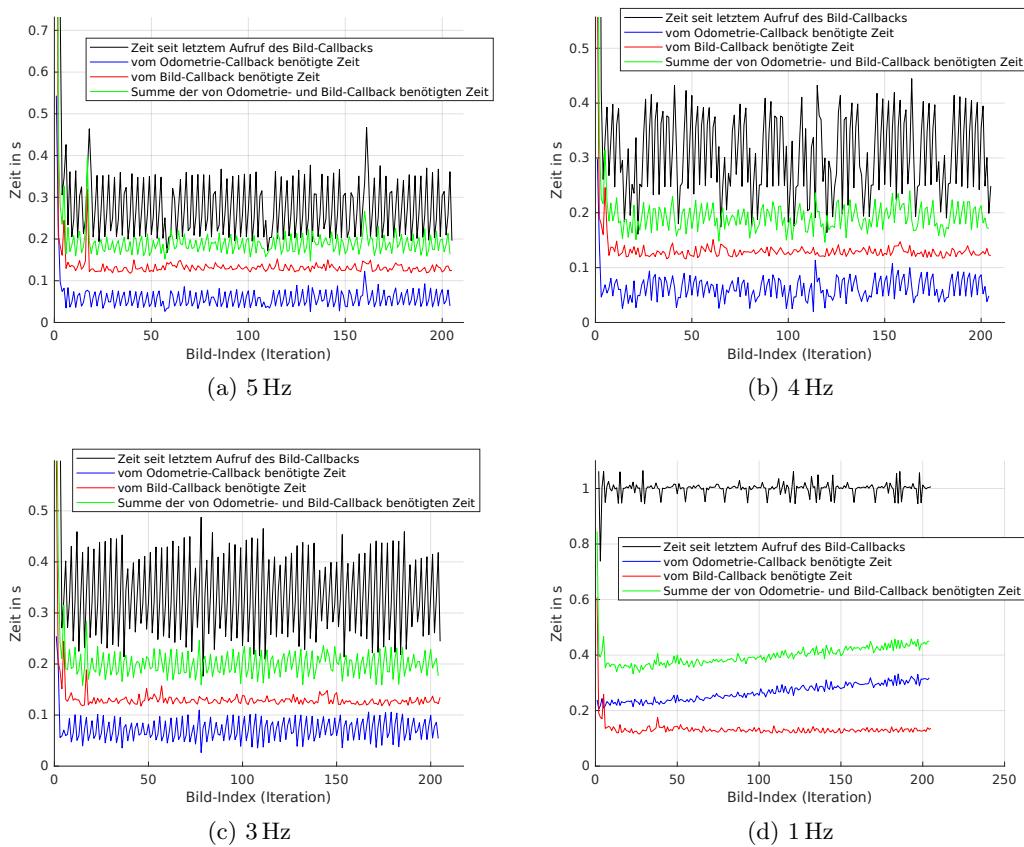


Abbildung 10.1.: Zeitbedarf aller Fahrspurverfolgungskomponenten bei 5 Hz, 4 Hz, 3 Hz und 1 Hz Bildfrequenz

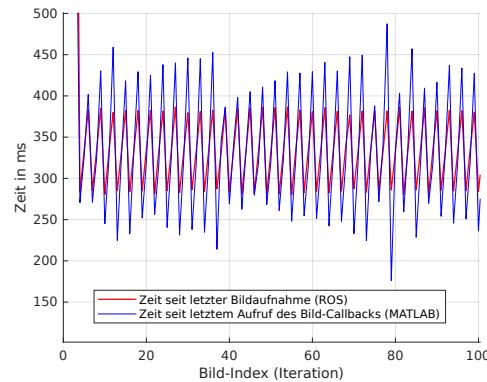


Abbildung 10.2.: Schwankungen Periodendauer Bildaufnahme/Bild-Callback bei 3 Hz Bildfrequenz

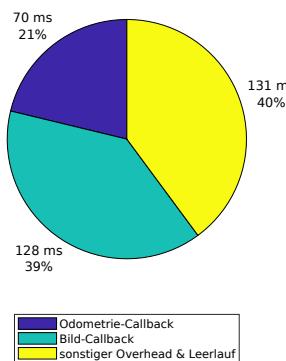
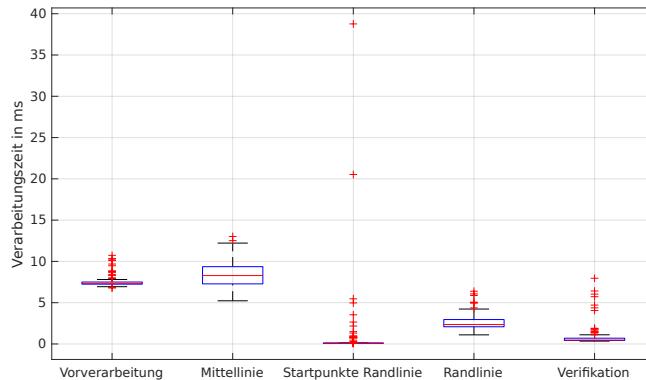
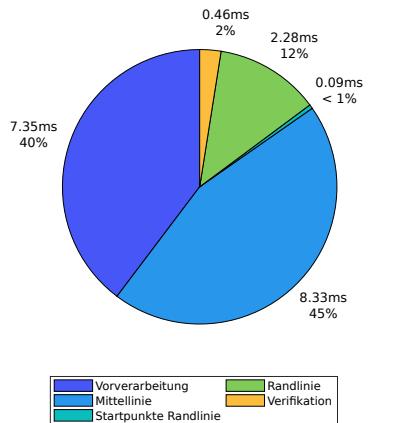


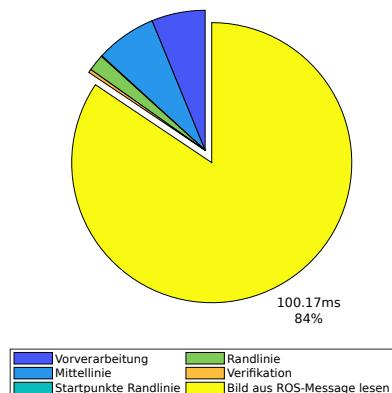
Abbildung 10.3.: Zeitbedarf aller Fahrspurverfolgungskomponenten pro Aufnahme bei 3 Hz Bildfrequenz; Median aller Messwerte einer Runde der Teststrecke



(a) Boxplot



(b) Tortendiagramm; Median aller Messwerte einer Runde



(c) Tortendiagramm; Median aller Messwerte einer Runde; mit Dekompression des Bildes

Abbildung 10.4.: Zeitbedarf aller Fahrspurerkennungskomponenten bei 3 Hz Bildfrequenz

in eine von MATLAB prozessierbare Matrix benötigt. Eine genauere Untersuchung mit dem MATLAB-Profiler zeigte, dass in diesem Schritt eine Dekompression des im JPEG-Format übertragenen Bildes den wesentlichen Anteil dieses Zeitbedarfs verursacht. Eine unkomprimierte Übertragung würde jedoch ein vielfaches dieser Dauer in Anspruch nehmen.

Der eigens entwickelte Anteil der Fahrspurerkennung benötigt im Median einer Runde lediglich 18.5 ms, die genaue Aufteilung kann in Abb. 10.4b sowie 10.4a nachvollzogen werden. Die meiste Zeit wird für Bildvorverarbeitung und Mittellinienerkennung benötigt. Da diese Funktionen zum Großteil auf vorgegebenen, komplexen und bereits maschinennah implementierten MATLAB-Funktionen basieren, kann hier nur unter Nutzung anderer Software eine Optimierung stattfinden. Die zum Großteil aus elementaren Funktionen aufgebaute Erkennung & Verifikation der Randlinie läuft schon hinreichend schnell ab, sodass hier trotz der vielen verwendeten, in MATLAB rechenzeitintensiven Schleifen keine effizientere Alternative gesucht werden muss.

**Startpunktfindung** Wie in Abb. 10.4b zu sehen, verstreicht im Normalfall bei der Feststellung der Startpunkte zur Erkennung der seitlichen Fahrbahnmarkierungen kaum Zeit. Wird jedoch kein Mittellinienelement nahe genug vor dem Fahrzeug gefunden, so gestaltet sich dieser Prozess aufwendiger (siehe Abschnitt 8.2.1). Abbildung 10.5 gibt ein besseres Bild der Messwerte wieder als Diagramm 10.4b bzw. 10.4a, da hier nicht nur der Median aller Messwerte gebildet, sondern vor der Erstellung des Boxplots die Daten nach der Methode zur Startpunktbestimmung sortiert wurden. Die Erwartung, dass bei Nutzung der mittigen Fahrbahnmarkierung zur Startpunktgenerierung kaum Zeit vergeht, wurde durch den über diese Kategorie gebildeten Median von 0.09 ms bestätigt. Jedoch ist auch die von der eindimensionalen Hough-Transformation im Median benötigte Laufzeit von 1.32 ms vertretbar. Die in Abbildung 10.5 sichtbaren Ausreißer entstehen durch das erstmalige Ausführen der Funktion, welche nicht während der im Boxplot unberücksichtigten „Warmlaufphase“ geschieht. Da nur einmal fix im Bildkoordinatensystem liegende Punkte genutzt werden mussten und dieser Fall nur nach Fehlschlag der eindimensionalen Hough-Transformation auftritt, wird dieser Messwert nur als Median ohne Box etc. in Abb. 10.5 dargestellt und bewegt sich mit 2.17 ms im Bereich dieser Methode.

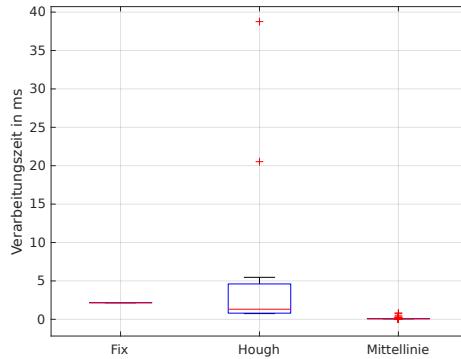


Abbildung 10.5.: Zeitbedarf der Startpunktfindungsalgorithmen für die Erkennung der seitlichen Fahrbahnmarkierungen bei 3 Hz Bildfrequenz

**Übertragungszeit** Durch JPEG-Kompression mit einer Qualitätseinstellung von 20% wurde der Bildtransfer ohne großen Informationsverlust auf 77.2 ms im Median beschleunigt (s. Abb. 10.6). Diese Übertragungszeit stellt immer noch eine erhebliche Totzeit dar, lässt sich aufgrund der hohen benötigten Rohdatenauflösung von 1080x1080 Pixeln ohne grundsätzliche Änderung des verwendeten Hardwareaufbaus und der Softwarestruktur jedoch nicht weiter verringern. Die Streuung der Messwerte bis hin zu 183 ms spiegelt die Probleme der Nutzung einer WLAN-Verbindung, welche zu unvorhersehbaren Latenzen neigt, wieder. Die angesprochene Totzeit schlägt sich jedoch nicht direkt auf das Fahrverhalten des Modellautos nieder, da die Bildverarbeitung wie in Passage 5.2.2 beschrieben getrennt von der Regelung des Roboters abläuft.

## 10.4. Qualität der Weltkarte **K**

Alle für die Regelung notwendigen Informationen über den Verlauf der Straßenmarkierungen stammen aus der parallel angelegten Karte. Damit eine stabile Navigation möglich ist, muss ihre Reliabilität vorausgesetzt sein. Die Authentizität der Kartenpunkte hängt wiederum von der Zuverlässigkeit der Pose ab. Da IMU und Encoder fehlerbehaftet sind, ist eine Abweichung der ermittelten zur realen Pose zu erwarten.

Um die tatsächliche Genauigkeit der Pose und Weltkarte anschaulich darzustellen, wurden in Abbildung 10.7a die geplotteten Karten- und Posendaten maßstabsgerecht

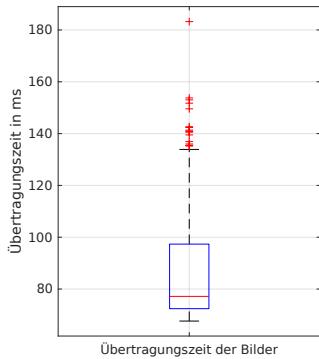


Abbildung 10.6.: Übertragungszeit der Aufnahmen bei 3 Hz Bildfrequenz

über die Entwurfszeichnung gelegt. Trotz sich theoretisch aufsummierender Fehler ist die Abweichung der End- zur Startpose nach einer Runde erstaunlich gering.

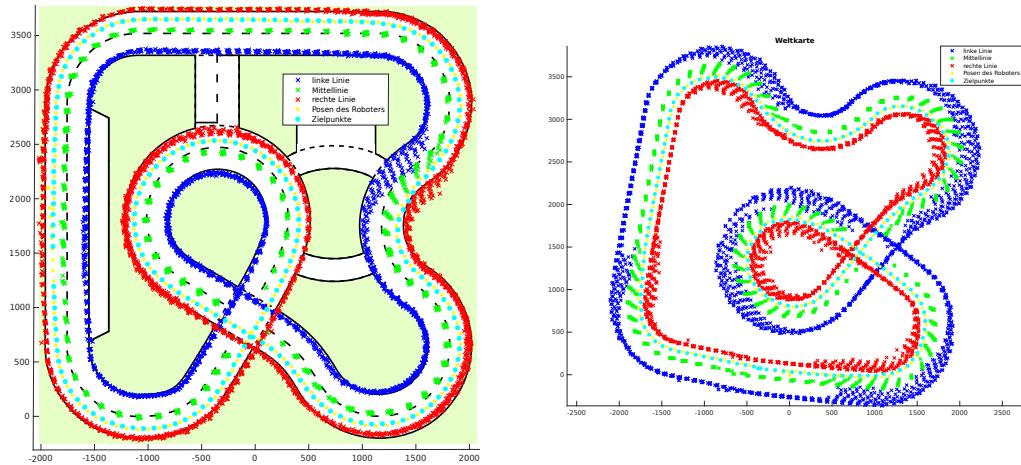
Durch die Kennzeichnung von rechter und linker Randlinie ist die Fahrtrichtung zu erkennen. In der Rechtskurve fällt auf, dass eine zu starke Krümmung der Linien erkannt wurde. Die Ursache dieses Fehlers liegt in der noch leider insuffizienten Bildentzerrung. Dies führt dazu, dass der Roboter die steileren Rechtskurven etwas schneidet. Nichts desto trotz gelingt die Fahrt auf dem Parcours auch in entgegengesetzter Richtung, wie Abbildung 10.7b zeigt.

## 10.5. Geschwindigkeit

Nachdem eine passende Einstellung der Regel- und Bildverarbeitungsfrequenz gefunden wurde, soll nun das Fahren mit höheren Geschwindigkeiten betrachtet werden. Neben der Sichtprüfung der Funktionalität der Fahrspurverfolgung wurden die Anzahl der zur Zielpunktbestimmung genutzten Punkte pro Regelungsiteration und der prozentuale Anteil an nicht bestimmten Zielpunkten zur Untersuchung herangezogen.

Zur besseren Vergleichbarkeit der Messergebnisse wurden die Daten für je eine entgegen des Uhrzeigersinnes gefahrene Runde mit 3 Hz Bildverarbeitungsfrequenz aufgenommen.

Abbildung 10.8a zeigt exemplarisch für die Geschwindigkeit  $v = 0.2 \text{ m s}^{-1}$  die Anzahl jeweiliger Linienpunkttypen zur Zielpunktberechnung. Es fällt sogleich auf, dass die Mittellinie einen kleineren Einfluss nimmt. Da pro Strich und Bild nur ein Punkt



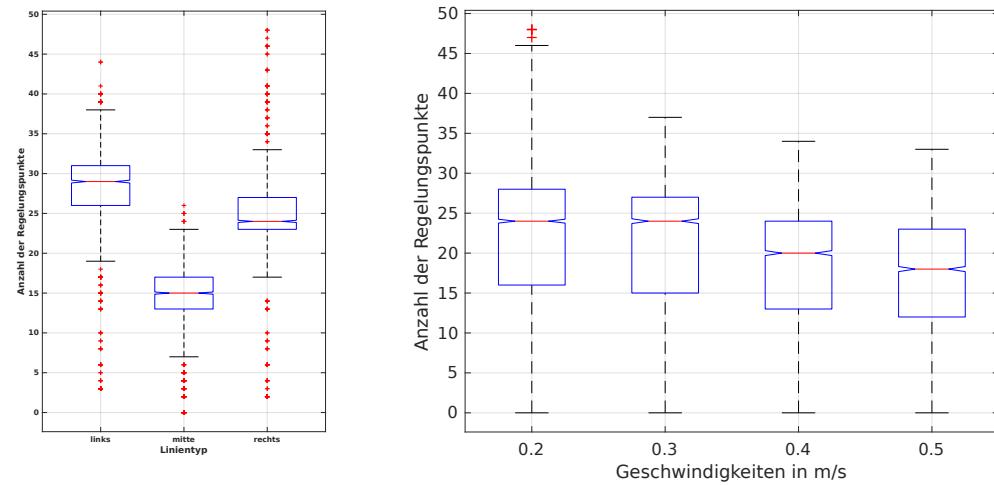
(a) Kombination des Plots der durch Messungen entstandenen Weltkarte mit der realen Entwurfszeichnung in passender Größe als Qualitätsnachweis  
 (b) Plot der durch Messungen entstandenen Weltkarte nach einer gefahrenen Runde im Uhrzeigersinn

Abbildung 10.7.: Darstellungen aufgenommener Karten des Szenarios zur Qualitätsuntersuchung

in der Weltkarte eingetragen wird, ist deren Punkteanzahl geringer. Wie unter 3 in Kapitel 9.2.1 beschrieben, werden die Regelungspunkte unter anderem nach ihrer x-Koordinate ausgewählt. In einer Linkskurve fallen dadurch mehr Punkte der linken Randlinie in das Suchfenster. Da die Strecke in der befahrenen Richtung vorrangig Linkskurven aufweist, werden im Mittel mehr linke als rechte Randpunkte zur Zielpunktberechnung herangezogen.

Betrachten wir die Gesamtzahl der zur Regelung verwendeten Punkte, ist wie in Abbildung 10.8b ein Abfallen der Anzahl mit ansteigender Geschwindigkeit des Fahrzeugs zu beobachten. Wenn das Auto schneller fährt, liegt ein Punkt in der Karte umso eher außerhalb des Bereiches, aus dem die Regelungspunkte genutzt werden. Bei gleichbleibender Bildfrequenz wird zwischen jedem Bild eine größere Strecke zurückgelegt, sodass die Dichte der Weltkartenpunkte geringer wird. Der dargestellte Boxplot erfüllt also die Erwartungen.

Die mit unserer Implementierung erreichbare Höchstgeschwindigkeit sollte folglich dann erreicht sein, wenn der mobile Roboter fast ausschließlich nach den Informationen eines Bildes regelt oder sogar zwischenzeitlich keine Zielpunkte mangels Rege-



(a) Anzahl der jeweiligen zur Regelung genutzten Punkte aus der Weltkarte bei  $v = 0.2 \text{ m s}^{-1}$   
 (b) Anzahl der zur Regelung verwendeten Kartenpunkte zu verschiedenen Geschwindigkeiten

Abbildung 10.8.: Boxplots zur Menge der Punkte, welche in einem Regelungsintervall aus der Weltkarte entnommen und zur Zielpunktberechnung verwendet werden

lungspunkten bestimmen kann. Mit anderen Worten: Wenn sich das Auto zwischen zwei Aufnahmen bis zum Sichtende des ersten Bildes bewegt, wie der Riverflow-Algorithmus Punkte erkennen konnte, ist die Maximalgeschwindigkeit beinahe erreicht. Ein Indiz dafür ist neben der visuellen Auffälligkeit des Fahrspurverlasses die starke Zunahme der nicht gefundenen Zielpunkte.

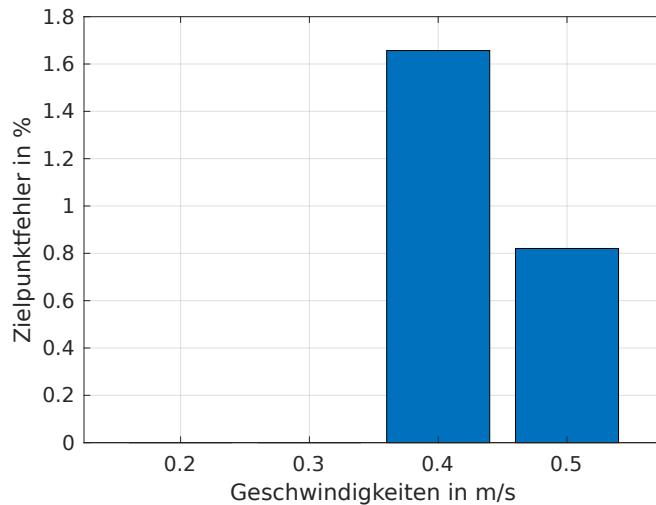


Abbildung 10.9.: Anteil der unbestimmten Zielpunkte einer Runde auf dem Parcours mit verschiedenen Geschwindigkeiten

Was in Abbildung 10.9 zu sehen ist, war ebenfalls während des Fahrversuches am Auto zu beobachten. Ab einer Geschwindigkeit von  $v = 0.4 \text{ m s}^{-1}$  fuhr das Fahrzeug instabiler und die Latenzzeit der Regelung war dahingehend zu bemerken, dass in Kurven meist etwas zu spät und kurz darauf zur Korrektur zu viel eingelenkt wurde. Dies führte zu einer schwingenden Fahrweise (das Auto fuhr „Schlängellinien“).

Die begrenzende Komponente für die Geschwindigkeit ist hier neben der Bildfrequenz allerdings auch das Getriebe. Durch dessen kurze Übersetzung beträgt die Geschwindigkeit im Maximum 4500 tics pro Sekunde, was in etwa  $0.43 \text{ m s}^{-1}$  entspricht. Die im Plot bezeichneten  $0.5 \text{ m s}^{-1}$  dürften folglich nicht genau stimmen, auch wenn dies der eingestellte Wert des Geschwindigkeitsparameters war. Sie geben die Daten zu der dem ersten Auto maximal physikalisch möglichen Geschwindigkeit an.

Ergänzend muss hier erwähnt werden, dass bereits ein zweites Fahrzeug mit ca.

doppelt so langer Übersetzung existiert. Jedoch misslang eine Testfahrt mit unveränderten Parametereinstellungen bei höheren Geschwindigkeiten. Auch hier zeigte sich die Grenze an der  $0.5 \text{ m s}^{-1}$ -Marke.

Dass diese Geschwindigkeit auch die Grenze der Stabilität des Algorithmus war, zeigte ein „Dauertest“ von zehn Runden. Bis zu einer Geschwindigkeit von  $0.3 \text{ m s}^{-1}$  führte der Versuch zu keinen nennenswerten Problemen in der Fahrspurverfolgung. Auch wenn während der schnelleren  $0.4 \text{ m s}^{-1}$ -Fahrt zehn Runden erfolgreich abgefahren wurden, kam hier das Fahrzeug mitunter stark ins Schwingen und befand sich kurz davor, den Fahrstreifen zu verlieren. Mit der höchstmöglichen Geschwindigkeit wurde der Dauertest nicht bestanden, da das Auto schon nach zwei Runden von der Straße abkam.

## 10.6. Hinzufügen von Objekten zum Testszenario M

Im realen Straßenverkehr befinden sich in der Regel mehr Objekte im Sichtbereich des Fahrers bzw. der Kamera als die bis jetzt im Parcours alleinig vorhandenen Fahrbahnmarkierungen. Diese Gegenstände können für die Trajektorieplanung relevant sein (Hindernisse) oder in keinem Zusammenhang damit stehen (Objekte abseits der Straße). Da der bis jetzt implementierte Fahrspurverfolgungsalgorithmus keine Hinderniserkennung und -vermeidung besitzt, kann hier nur auf Gegenstände neben der Fahrspur eingegangen werden. In diesem Abschnitt sollen Untersuchungen stattfinden, ob die Erkennung des Straßenverlaufs auch bei Hinzufügen weiterer Gegenstände zum Testszenario noch zufriedenstellend abläuft.

**Betrachtungen zur Bildvorverarbeitung** Vor der Durchführung von Testfahrten muss die Frage gestellt werden, wann dem Parcours hinzugefügte Elemente einen Einfluss auf die Fahrspurerkennung haben können. Da die Filterung und anschließende Binarisierung des entzerrten Graustufenbildes den Ausgangspunkt für daran folgende Programmkomponenten darstellen, muss das Objekt die „Hürde“ dieser Informationsreduktion überwinden können. Konkret bedeutet dies, dass sein Kontrast im Vergleich zum umliegenden Szenario groß genug sein muss, um als schwertüberschreitende Filterantwort des Kantendetektors im Binärbild aufzutreten oder als relevantes Maxima auf einer Scanline gefunden zu werden (s. Abschnitt 6.3 bzw. 7.2.2). Als Vorversuch wurden farbige Blätter (Abb. 10.10a) verschiedener Hellig-

keit (Abb. 10.10b) im Testszenario platziert. Abbildung 10.10c zeigt wie erwartet, dass nur sehr dunkle Farben eine ausreichend große Filterantwort hervorrufen, welche im Binärbild (Abb. 10.10d) als Kante vermerkt wird. Auffällig ist auch, dass die Eckpunkte der Blätter eine besonders große Filterantwort bewirken, die eigentlichen Kanten aber nur beim roten (teilweise) und schwarzen Blatt erkannt werden.

**Objekte neben der Fahrspur** Es wurden einige Quadern, in den Farben des in Abschnitt 10.6 untersuchten Papiers, im Parcours positioniert (s. Abb. 10.11). Hierbei wurde bei 3 Fahrten der Abstand zur seitlichen Fahrbahnmarkierung von 16 cm (größer der Länge einer Scanline) über 4 cm (kleiner der Länge einer Scanline) hin zu 0 cm verändert.

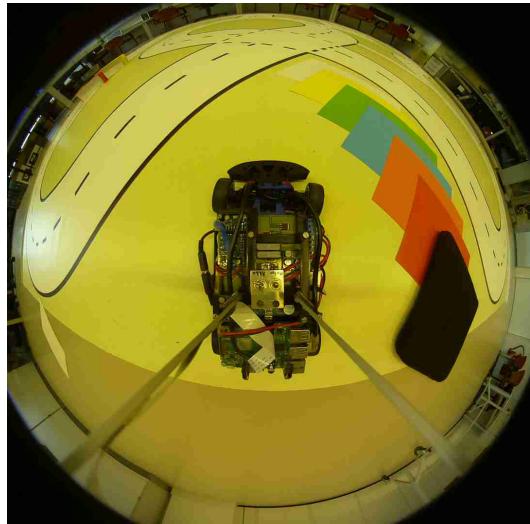
Können die Scanlines die Kanten eines solchen Objektes nicht erreichen, so ist kein Einfluss auf die Erkennung der Randlinien zu erwarten. Ist der Kontrast eines Quaders zur hellgrünen Grundfarbe des Testszenarios zu gering, so ist eine Fehldetektion aufgrund dessen ebenfalls ausgeschlossen. Eventuelle Beeinträchtigungen des Riverflow-Algorithmus sind somit nur bei den Versuchen mit 4 cm und 0 cm Abstand bei dunklen Farben der „Störobjekte“ zu erwarten.

Der Roboter fuhr jedoch auch in diesen Konstellationen fehlerfrei die Runde. Es konnten bei allen Testfahrten keine falsch in der Weltkarte eingetragenen Punkte identifiziert werden.

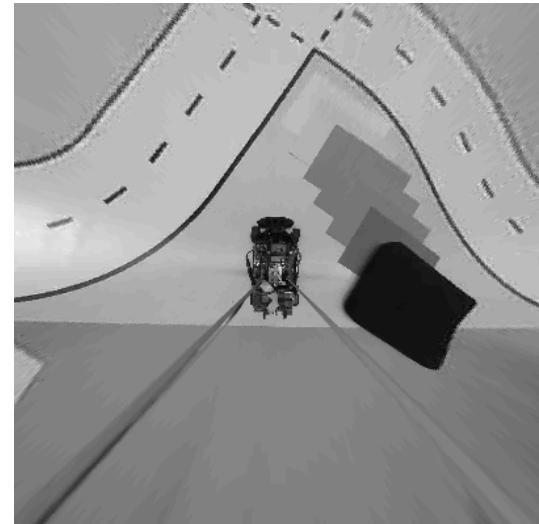
Diese Robustheit gegenüber den hinzugefügten Quadern wurde auf die Verifikation der Punkte nach der Erkennung zurückgeführt, somit soll nun noch die Performanz des Algorithmus ohne diese Komponente betrachtet werden. Charakteristisch für eine Beeinträchtigung an dieser Stelle wäre das Anlegen vieler Hypothesen (möglicher Verläufe der Randlinien), da auf einer Scanline mehrere potentielle zentral auf einer Fahrbahnmarkierung gelegenen Punkte gefunden werden (s. Abschnitt 8.2.2).

Abbildung 10.12 bestätigt diese Vermutung, es ist ein häufigeres Auftreten mehrerer Hypothesen für die seitlichen Fahrbahnmarkierungen zu beobachten, wenn diese in die von den Scanlines aufgespannten RoI's platziert werden. Beispiele für diese Fehlerkennungen sind in Abbildung 10.13 zu sehen.

Ein weiterer Grund für die Detektion mehrerer möglicher Verläufe der Randlinie stellt, wie in Abbildung 10.13d zu sehen, die Erkennung schon immer im Testszenario vorhandener, bis jetzt nicht weiter berücksichtigter Kanten dar. Die Ränder der PVC-Plane, auf welche die Strecke gedruckt wurde oder die zum Beschweren dieser



(a) roh



(b) entzerrt



(c) gefiltert



(d) binarisiert

Abbildung 10.10.: Vorversuch Kontraste zusätzlicher Objekte im Testszenario



Abbildung 10.11.: Versuchsaufbau farbige Quader im Testszenario

genutzten Alu-Profile sind häufig die Ursache solcher Fehldetektionen.

Die unter Nachteile in Abschnitt 10.1 vermutete Verlängerung der Bearbeitungszeit konnte bestätigt werden, Abb. 10.14 stellt den Zusammenhang zwischen der Anzahl verfolgter Hypothesen und der damit verbundenen Dauer der Detektion seitlicher Fahrbahnmarkierungen dar.

**Objekte in der Gegenspur** Da, bedingt durch die Verifikation, erkannte Fahrbahnmarkierungen nur in die Karte eingetragen werden, wenn sie zu mindestens einer weiteren erkannten Linie passen, muss für eine Fehlerkennung diese Bedingung auch durch die platzierten „Störobjekte“ erfüllt werden. Da sich das Fahrzeug nur auf der rechten Straßenseite fortbewegt, wurde der Versuchsaufbau in Abb. 10.15 gewählt, welcher diesem Kriterium nachkommt.

Die Fehldetektion der linken Fahrbahnmarkierung konnte somit provoziert werden. Da die richtig erkannte rechte Randlinie jedoch mehr Punkte generierte und anhand der Mittellinie verifiziert wurde, sind diese zwei Linien noch korrekt in die Weltkarte eingetragen (s. Abb. 10.16c und 10.16a). Das Modellauto fuhr ein wenig rechts der Ideallinie, kam aber nicht stark vom Kurs ab. Erst durch Abdecken der Mittellinie (s. Abb. 10.16d und 10.16b) und die somit fehlende Validierungsmöglichkeit der richtig erkannten rechten Fahrbahnmarkierung konnte der Roboter vom Kurs abgebracht werden und verließ die Straße völlig.

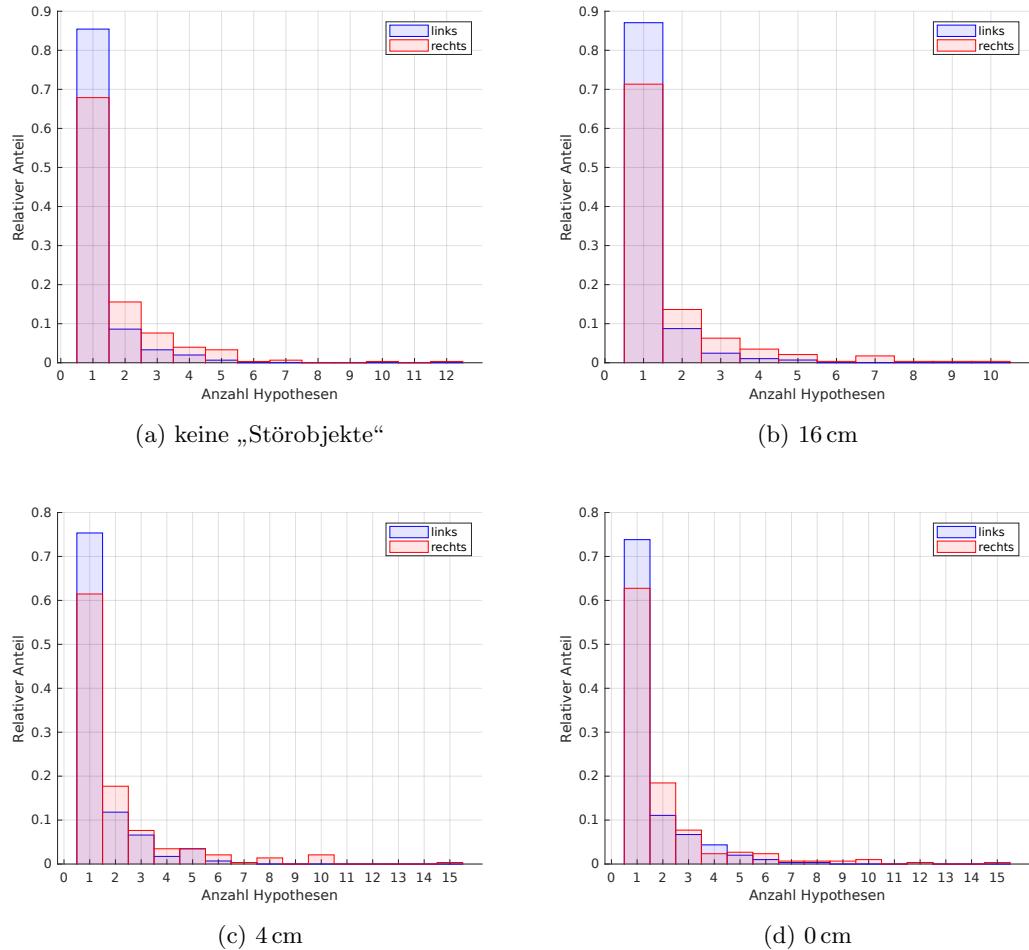


Abbildung 10.12.: Histogramme Hypothesenanzahl mit „Störobjekten“ im Testszenario; Variation des Abstandes von der Fahrspur

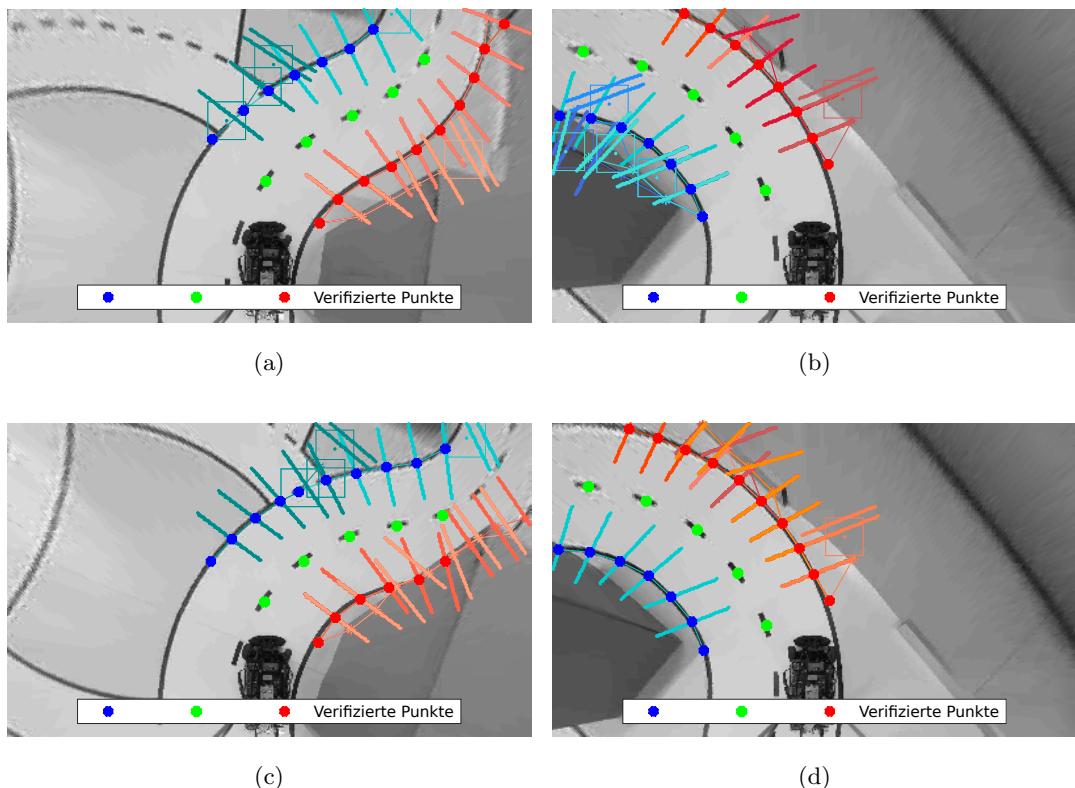


Abbildung 10.13.: Beispiele für die Erkennung mehrerer Hypothesen mit „Störobjekten“ im Testszenario

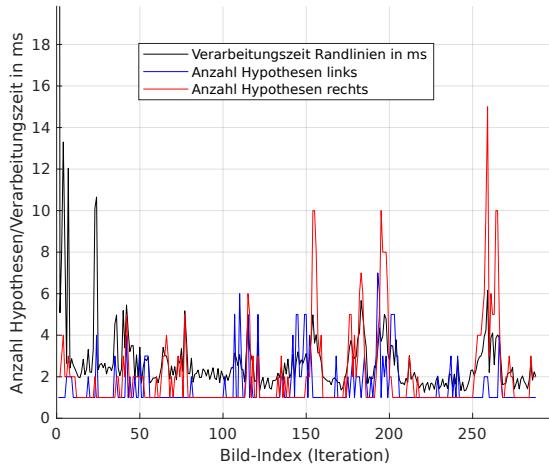


Abbildung 10.14.: Zusammenhang Anzahl verfolgter Hypothesen & Dauer der Detektion seitlicher Fahrbahnmarkierungen; 4 cm Abstand Quader zu Fahrspur

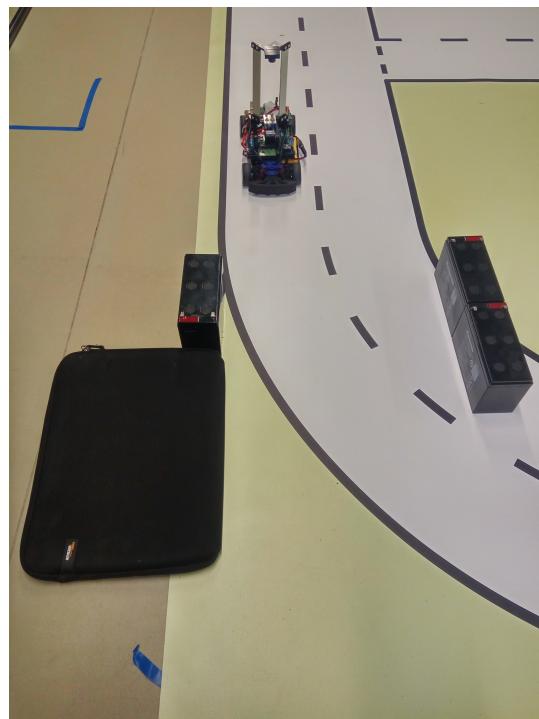


Abbildung 10.15.: Versuchsaufbau mit „Störobjekt“ auf der Gegenfahrspur

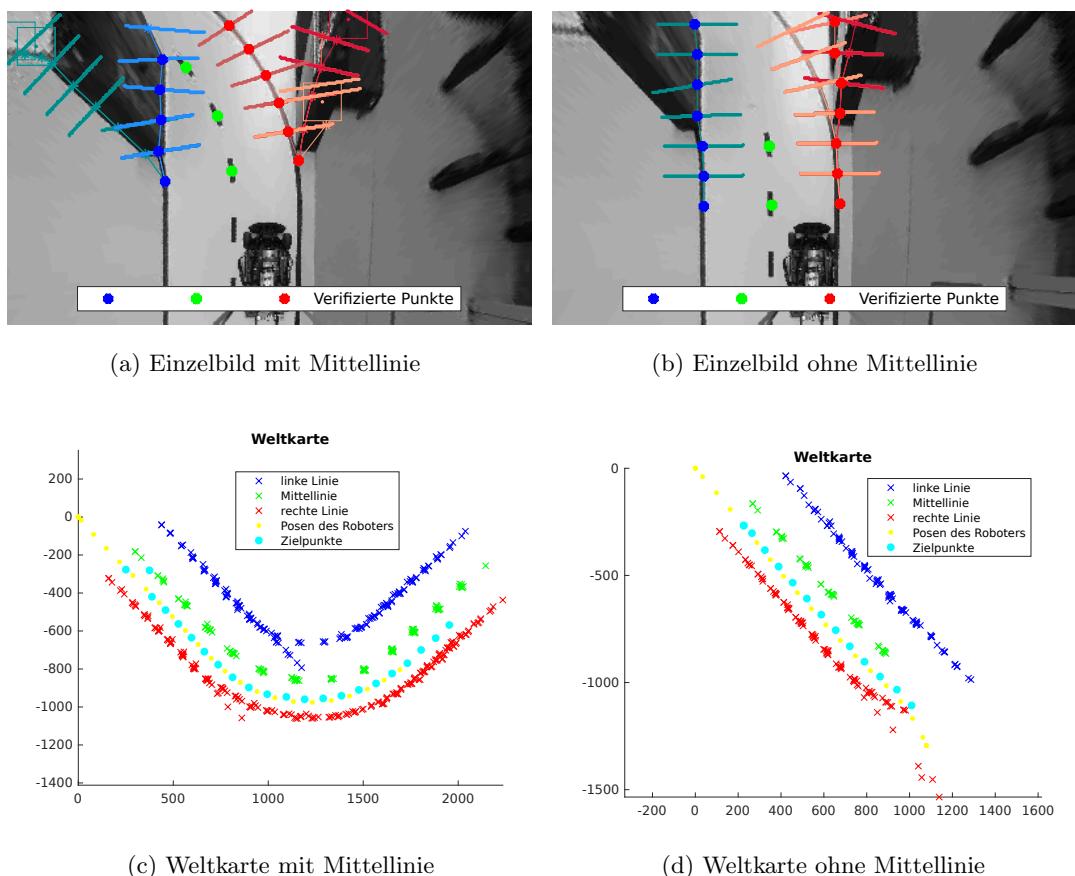


Abbildung 10.16.: Versuch mit „Störobjekt“ auf der Gegenfahrspur

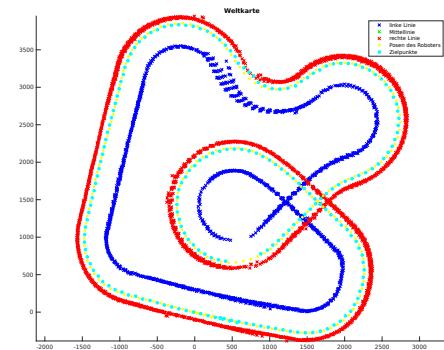
## 10.7. Fahren mit weniger Informationen K

Unsere Linienerkennung besteht im wesentlichen aus den zwei fusionierten Komponenten Mittelstrich- und Randlinienerkennung, was die Robustheit unseres Algorithmus wesentlich erhöht. Um zu zeigen, dass diese nicht nur theoretisch unabhängig voneinander funktionieren, wurden zum Test je einmal die Mittellinie und die beiden Randlinien mit weißem Papier verdeckt.

**Fahren ohne Mittellinie** Da in diesem Test keine Mittellinienpunkte gefunden wurden, konnten die Startpunkte für den Riverflow-Algorithmus nur wie in Kapitel 8.2.1 beschrieben mittels der vereinfachten Hough-Transformation oder der Annahme fixer Punkte bestimmt werden. Abbildung 10.17 zeigt den Versuchsaufbau und das Ergebnis der Karte nach einer erfolgreich gefahrenen Runde ohne Mittellinie.



(a) Versuchsaufbau ohne Mittellinie



(b) Weltkarte ohne Mittellinie

Abbildung 10.17.: Testfahrt mit durch weißes Papier abgedeckten Mittelstrichen

**Probleme** Damit die gesamte Runde ohne Mittellinie gemeistert werden konnte, wurden während des Versuchs bestimmte Parameter angepasst. Stand der Roboter beispielsweise zu schräg in der Fahrspur, geriet die Randlinie schnell aus dem Bereich, in dem die vereinfachte Hough-Transformation ausgeführt wird. Deshalb wurde die Breite dieses Bildausschnittes vergrößert. Dies führt allerdings zu einem Nachteil, welcher der Grund für die anfangs schmalere Einstellung war und in Abbildung 10.18

dargestellt ist. Da der Hough-Algorithmus nur nach den größten Maxima sucht (siehe Abschnitt 4.6.1), erkennt er mitunter bei zu breiter Fenstereinstellung benachbarte Markierungen parallel verlaufender Straßen. Die in Kapitel 8.3 erläuterte Verifikation verhindert erfreulicherweise die Aufnahme der falschen Punkte in die Karte. Da der Roboter jedoch auch keine Richtigen erkannte, erhält er in diesem Fall keine neuen Informationen zur Zielpunktfindung.

Die Startpunktgewinnung mittels Mittellinie bietet darüber hinaus den großen Vorteil, dass die Richtung des Startpunktvektors bekannt ist. Da diese Information hier fehlt, zeigt der Verschiebungsvektor in x-Richtung des Roboterkoordinatensystems. In engen Kurven führt das in ungünstigen Fällen wie in Abbildung 10.18 zum Abbruch, da die nächste Scanlinie die Fahrbahnmarkierung nicht mehr schneidet. Das ist der Grund für die große Lücke in der linken Linie der in Abbildung 10.17b dargestellten Weltkarte.

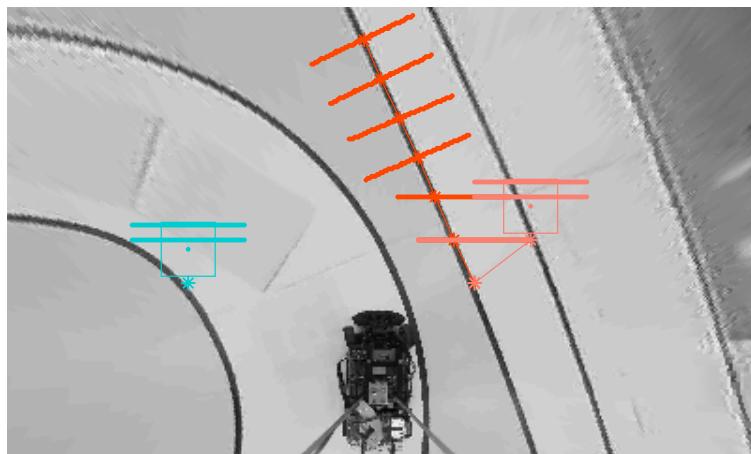
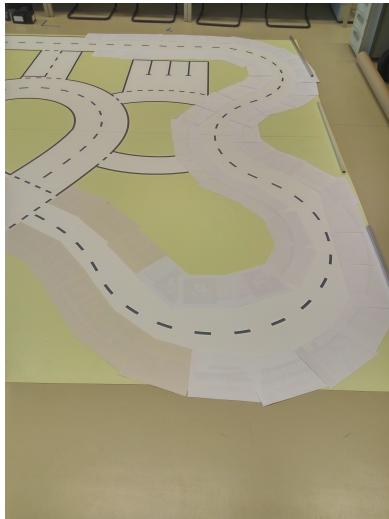


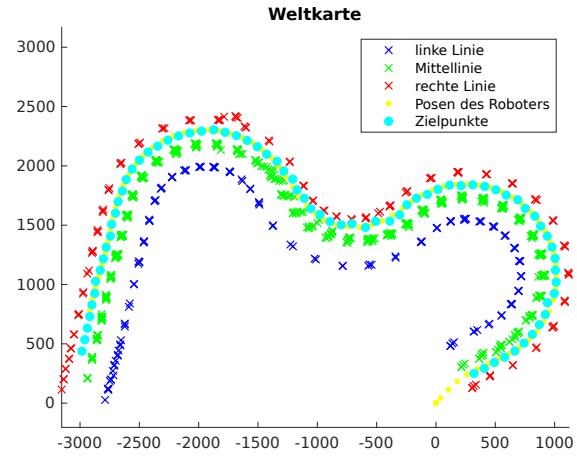
Abbildung 10.18.: Ungünstige Position für die mit Hough initialisierte Randlinienerkennung

**Fahren ohne Randlinien** Die bisher gute Qualität der binarisierten Bilder ermöglicht eine zuverlässige Mittellinienerkennung. Da diese von keinerlei Startpunkten oder Ähnlichem abhängt, stellen auch enge Kurven oder benachbarte Straßenabschnitte kein Problem dar. Der Versuchsaufbau und die zugehörig aufgenommene Weltkarte sind in Abbildung 10.19 dargestellt.

Im Plot 10.19b fällt auf, dass trotz Abdeckung scheinbar einige Randlinienpunkte



(a) Versuchsaufbau ohne Randlinien



(b) Weltkarte ohne Randlinien

Abbildung 10.19.: Testfahrt mit durch weißes Papier abgedeckte Randlinien

erkannt wurden. Mithilfe des in Abbildung 10.20 dargestellten Beispielplots lässt sich die Erklärung finden.

Ausgehend von der ersten mittleren Fahrbahnmarkierung werden die Startpunkte für die Randlinienerkennung bestimmt. Trotz fehlender Randlinienpunkte wurden die Startpunkte verifiziert und in die Karte aufgenommen. Da sie jedoch einzig den um die Fahrspurbreite verschobenen Mittellinienspunkt abbilden, stellen sie keinen informativen Mehrwert dar.

**Schlussfolgerungen** Wir haben demonstriert, dass die Rand- und Mittellinienerkennung einzeln funktioniert und sich gegenseitig absichert. Das TUCar ist nicht auf permanentes Vorhandensein von Mittel-/Randlinien oder deren fehlerfreie Erkennung angewiesen. Die Zielpunktbestimmung als Voraussetzung für die Regelung ist somit gewährleistet, wenn mindestens die Mittellinie oder beide Randlinien erkannt wurden. Diese Sicherheit ist durch die Schwächen der jeweiligen Verfahren sinnvoll und notwendig.

An der großen Kreuzung existiert beispielsweise keine mittlere Fahrbahnmarkierung, was die Randlinienerkennung notwendig macht. Gleichzeitig besitzt der Riverflow-

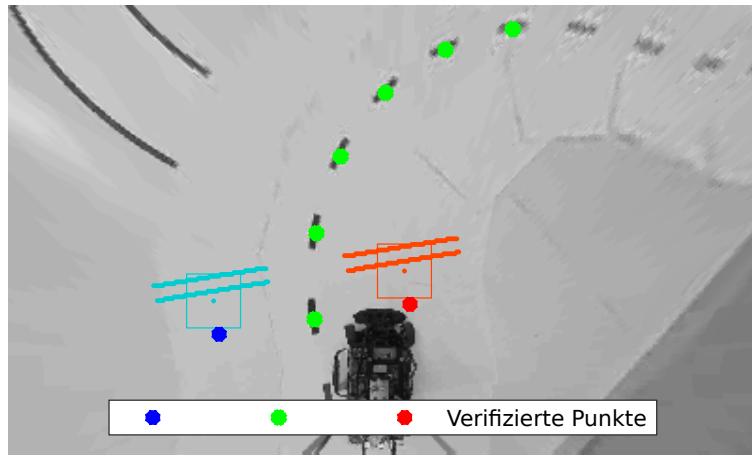


Abbildung 10.20.: Punkterkennung bei abgedeckten Randlinien

Algorithmus den Nachteil, an ausreichend großen Lücken im Markierungsverlauf abzubrechen. Dieser Fall ist demonstrativ in Abbildung 10.21a dargestellt.

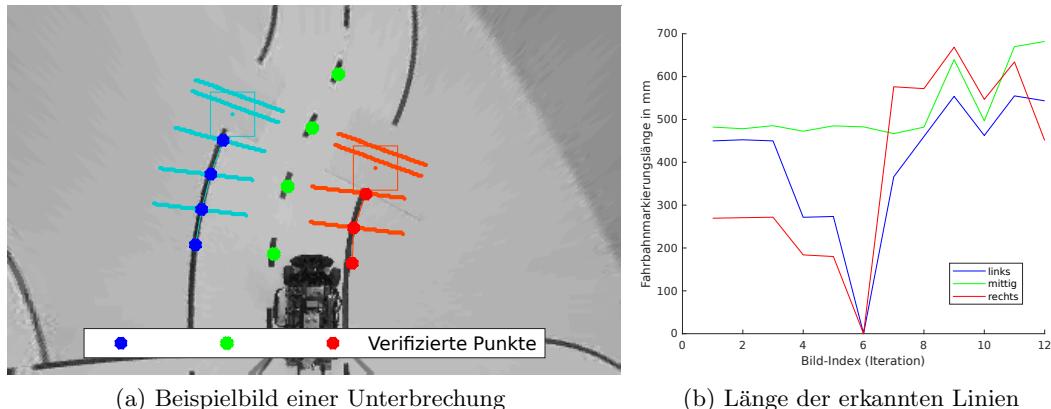


Abbildung 10.21.: Demonstration der Schwäche des Riverflow bei größeren Unterbrechungen in der Randlinie

Die durch ein A4-Blatt unterbrochene Randlinie wird im folgenden Verlauf nicht länger erkannt. Dass die Linie in den vorherigen Bilditerationen ebenfalls nur bis zu jener Unterbrechung detektiert wurde, zeigt der Plot in Abbildung 10.21b durch die mit jedem Bildindex abnehmende Linienlänge. Die Fähigkeit, mittels Randlinienerkennung vorauszuschauen, ist somit bis zum Passieren dieser Unregelmäßigkeit nicht

---

gegeben. Durch die möglich fehlerhafte Erkennung eines Mittellinienpunktes werden überdies ebenfalls alle nachfolgenden Mittellinienpunkte ignoriert, da auch hier die Verifikation des nächsten Punktes von der des aktuellen abhängt.

## 10.8. Fazit

Dank der schon oft erwähnten hohen Bildqualität und dem starken Kontrast zwischen Markierungen und Hintergrund funktioniert die Fahrspurerkennung zufriedenstellend gut. Das unveränderte Szenario lässt die hier beschriebenen Fehler äußerst selten bis überhaupt nicht auftreten. Die jeweils souveräne Rand-und Mittellinienerkennung realisieren die Navigation des TUCars sehr robust und zuverlässig. Der die Geschwindigkeit begrenzende Flaschenhals ist vorrangig in der Übertragungszeit der Daten per W-LAN, anderen Latenzen und der dadurch begrenzten Bildverarbeitungsfrequenz zu finden.

## 11. Ausblick auf folgende Arbeiten **K**

Nachdem innerhalb dieser Gruppenarbeit eine stabile Fahrspurverfolgung mit Kartenaufbau in die Praxis umgesetzt wurde, liegen weiterhin viele Ideen zur Verbesserung, Ergänzung und zum Ausbau der Komplexität der Fähigkeiten des Autos vor. Da dieses Projekt der Anstoß für einen Praktikumsversuch sein soll, ist die Liste der Möglichkeiten lang. Denkbare und zukünftig erweiterbare Implementierungen für das TUCar wären z.B.:

- Anhalten an einer Kreuzung gleichrangiger Straßen
  - setzt eine Kreuzungserkennung voraus
  - Vorfahrt gewähren „rechts vor links“
  - Erkennen eines zweiten Fahrzeugs (z.B. mittels QR-Code)
- Abbiegen an Kreuzungen und Abzweigen
- Überholmanöver
- Hinderniserkennung mit Anhalten und/oder Ausweichen
- Einparken
  - in Längs - und Querparkplätze
  - Zwischen Kartons oder andere Autos bei Längsparkplätzen
  - vorwärts und rückwärts einparken bei Querparkplätzen
- Variation der Geschwindigkeit in Abhängigkeit von aufgestellten/aufgelegten Geschwindigkeitsbegrenzungen

# Literatur

- Aly, Mohamed (Juni 2008). „Real Time Detection of Lane Markers in Urban Streets“. en. In: *2008 IEEE Intelligent Vehicles Symposium*, S. 7–12. DOI: [10.1109/IVS.2008.4621152](https://doi.org/10.1109/IVS.2008.4621152). arXiv: [1411.7113](https://arxiv.org/abs/1411.7113).
- Corke, Peter (2017). *Robotics, Vision and Control*. en. New York, NY: Springer Berlin Heidelberg. ISBN: 978-3-319-54412-0.
- Drauschke, Clemens (Apr. 2016). „Echtzeitfähige Startpunktalgorithmen für kamera-basierte Fahrspur-, Kreuzungs- und Hindernisidentifikation“. de. Bachelorarbeit. Hamburg: Hochschule für Angewandte Wissenschaften Hamburg.
- Fischler, Martin A und Robert C Bolles (1981). „Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography“. In: *Communications of the ACM* 24.6, S. 381–395.
- Jähne, Bernd (2005). *Digitale Bildverarbeitung: mit 155 Übungsaufgaben*. de. 6., überarb. und erw. Auflage. OCLC: 76657118. Berlin: Springer. ISBN: 978-3-540-24999-3.
- Kalman, Rudolph Emil (1960). „A New Approach to Linear Filtering and Prediction Problems“. In: *Journal of basic Engineering* 82.1, S. 35–45.
- Kröger, Fabian (2015). „Das automatisierte Fahren im gesellschaftsgeschichtlichen und kulturwissenschaftlichen Kontext“. de. In: *Autonomes Fahren*. Hrsg. von Markus Maurer u. a. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 41–67. ISBN: 978-3-662-45853-2 978-3-662-45854-9. DOI: [10.1007/978-3-662-45854-9\\_3](https://doi.org/10.1007/978-3-662-45854-9_3).
- Kunze, Lars u. a. (2018). „Reading between the Lanes: Road Layout Reconstruction from Partially Segmented Scenes“. en. In: S. 4.
- Lim, King Hann, Kah Phooi Seng und Li-Minn Ang (2012). „River Flow Lane Detection and Kalman Filtering-Based B-Spline Lane Tracking“. en. In: *International Journal of Vehicular Technology* 2012, S. 1–10. ISSN: 1687-5702, 1687-5710. DOI: [10.1155/2012/465819](https://doi.org/10.1155/2012/465819).

- Marchthaler, Reiner und Sebastian Dingler (2017). *Kalman-Filter: Einführung in die Zustandsschätzung und ihre Anwendung für eingebettete Systeme*. de. Lehrbuch. OCLC: 971203245. Wiesbaden: Springer Vieweg. ISBN: 978-3-658-16727-1 978-3-658-16728-8.
- Narote, Sandipann P. u. a. (Jan. 2018). „A Review of Recent Advances in Lane Detection and Departure Warning System“. en. In: *Pattern Recognition* 73, S. 216–234. ISSN: 00313203. DOI: 10.1016/j.patcog.2017.08.014.
- Nikolai Chernov (Juni 2012). *MATLAB Codes for Fitting Ellipses, Circles, Lines*. <http://people.cas.uab.edu/~mosya/cl/MATLABcircle.html>.
- Nischwitz, Alfred, Hrsg. (2011). *Computergrafik*. de. 3., neu bearb. Aufl. Computergrafik und Bildverarbeitung Alfred Nischwitz ... ; Bd. 1. OCLC: 838820270. Wiesbaden: Vieweg + Teubner. ISBN: 978-3-8348-1304-6.
- Papula, Lothar (2012). *Mathematik für Ingenieure und Naturwissenschaftler Band 2*. de. 13., durchgesehene Auflage. Bd. 2. Mathematik für Ingenieure und Naturwissenschaftler Lothar Papula ; Band 2. Springer Vieweg. ISBN: ISBN 978-3-8348-1589-7.
- (2016). *Mathematik für Ingenieure und Naturwissenschaftler Band 3*. de. 7., überarbeitete und erweiterte Auflage. Bd. 3. Mathematik für Ingenieure und Naturwissenschaftler Lothar Papula ; Band 3. OCLC: 932693163. Wiesbaden: Springer Vieweg. ISBN: 978-3-658-11923-2 978-3-658-11924-9 978-3-8348-1227-8.
- Peters, Falko (Apr. 2009). „FPGA-basierte Bildverarbeitungspipeline zur Fahrspurerkennung eines autonomen Fahrzeugs“. Deutsch. Bachelorarbeit. Hamburg: Hochschule für Angewandte Wissenschaften Hamburg.
- Risack, R u. a. (1998). „Robust Lane Recognition Embedded in a Real Time Driver Assistance System“. en. In: *in Proc. IEEE IV*, S. 35–40.
- Rufli, Martin, Davide Scaramuzza und Roland Siegwart (2008). „Automatic Detection of Checkerboards on Blurred and Distorted Images“. In: *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference On*. IEEE, S. 3121–3126.
- Scaramuzza, Davide (2007). „Omnidirectional Vision: From Calibration to Root Motion Estimation“. PhD Thesis. ETH Zurich.
- Scaramuzza, Davide, Agostino Martinelli und Roland Siegwart (2006a). „A Flexible Technique for Accurate Omnidirectional Camera Calibration and Structure

- from Motion“. In: *Computer Vision Systems, 2006 ICVS'06. IEEE International Conference On.* IEEE, S. 45–45.
- Scaramuzza, Davide, Agostino Martinelli und Roland Siegwart (2006b). „A Toolbox for Easily Calibrating Omnidirectional Cameras“. In: *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference On.* IEEE, S. 5695–5701.
- Taubin, G. (Nov. 1991). „Estimation of Planar Curves, Surfaces, and Nonplanar Space Curves Defined by Implicit Equations with Applications to Edge and Range Image Segmentation“. In: *IEEE Transactions on Pattern Analysis & Machine Intelligence* 13, S. 1115–1138. ISSN: 0162-8828. DOI: [10.1109/34.103273](https://doi.org/10.1109/34.103273).
- Wikipedia (2018a). „Faltungsmatrix — Wikipedia, Die Freie Enzyklopädie“. In: *Wikipedia*. [Online; Stand 5. Oktober 2018].
- (2018b). „Schnittpunkt — Wikipedia, Die Freie Enzyklopädie“. In: *Wikipedia*. [Online; Stand 5. Oktober 2018].

# **Anhang A.**

## **Daten-DVD**

### **A.1. Digitale Version der Arbeit (PDF-Format)**

Die Bachelorarbeit befindet sich als PDF-Dokument *Bachelorarbeit* im Wurzelverzeichnis der Daten-DVD.

### **A.2. Quellcode**

In den Ordnern *Polynomisierte Fahrspurerkennung* und *Punktbasierte Fahrspurerkennung* befindet sich der Quellcode zu den in Kapiteln 7 und 8 differenzierten Fahrspurerkennungsmethoden. Da die grundsätzliche Vorgehensweise bei der Inbetriebnahme gleich ist, wurde lediglich ein PDF-Dokument *Anleitung* im Wurzelverzeichnis der DVD erstellt.

#### **A.2.1. Polynomisierte Fahrspurerkennung**

In den Verzeichnissen *Polynomisierte Fahrspurerkennung/Ransac* und *Polynomisierte Fahrspurerkennung/Kalman* ist der letzte Entwicklungsstand der in Abschnitt 7.1 sowie 7.2 dargelegten, jedoch letztendlich verworfenen Fahrspurdetektionskonzepten, festgehalten. Da nie eine selbstständige Fahrt im Testszenario realisiert wurde, kann mit diesem Code lediglich der Linienverlauf in Einzelbildern extrahiert, sowie eine Weltkarte erstellt werden. Dies kann durch händische Steuerung des Fahrzeugs, sowie auf Basis der im Unterordner *Punktbasierte Fahrspurerkennung/ROSbags* befindlichen ROS-Bag-Dateien geschehen.

### **A.2.2. Punktbasierte Fahrspurerkennung**

Im Verzeichnis *Punktbasierte Fahrspurerkennung* ist der final genutzte Ansatz zur Fahrspurerkennung inklusive funktionstüchtiger Regelung zu finden. Hier befindet sich auch ein Zip-Archiv *Debug-Daten.zip*, welches die Datenbasis des Kapitels 10 enthält.

### **A.3. gesicherte Internetquellen**

In Ordner *gesicherte Internetquellen* sind, nach Zitierschlüssel sortiert, alle referenzierten Websites gespeichert.

# **Selbstständigkeitserklärung**

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendete Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Chemnitz, 22. Oktober 2018

---

Daniel Käppler, Leopold Mauersberger