HAW HAMBURG

Bachelorarbeit

Nils Schönherr

Kamera-basierte Minimalautonomie

Fakultät Technik und Informatik
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

Nils Schönherr

Kamera-basierte Minimalautonomie

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Bachelor of Science Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Stephan Pareigis
Zweitgutachter: Prof. Dr.-Ing. Andreas Meisel

Eingereicht am: 01.07.2019

Nils Schönherr

Thema der Arbeit

Kamera-basierte Minimalautonomie

Stichworte

Kamera, Autonomie, Minimal, Mikrocontroller, ESP32, Autonomes Fahren, Simulation, Carolo-Cup

Kurzzusammenfassung

Diese Arbeit befasst sich mit dem Aufbau eines autonom fahrenden, minimalen Systems. Als Prozessor kommt ein Mikrocontroller zum Einsatz und als einziger Sensor wird eine Kamera verwendet. Nach der Hardwarebeschreibung, welche ein eigenes Design des Fahrwerks enthält, werden Softwaregrundlagen für Telemetrie und die Verwendung der Kamera beschrieben. Anschließend werden leichtgewichtige Algorithmen erläutert, welche die Bildverarbeitung auf dem Mikrocontroller mit eingeschränkten Ressourcen ermöglichen. Die Algorithmen werden abschließend in einer Simulation und in einem Livetest, der sich am Regelwerk des Carolo-Cups orientert, evaluiert.

Nils Schönherr

Title of Thesis

Camera-based Minimal Autonomy

Keywords

camera, autonomy, minimal, microcontroller, ESP32, autonomous driving, simulation, Carolo-Cup

Abstract

This thesis deals with the construction of an autonomous driving, minimal system. The used processor is a microcontroller and the only sensor is a camera. After describing

the hardware, which contains an own design of the chassis, the software basics for telemetry and the camera interface are described. Subsequently, lightweight algorithms are explained which allow image processing on the microcontroller with limited ressources. Finally, the alogrithms will be evaluated in a simulation and in a live test based on the regulations of the Carolo-Cup.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

BLOB Binary Large Object.

CAD Computer-Aided Design.

DMA Direct Memory Access.

ESP-IDF Espressif IoT Development Framework.

FDM Fused Deposition Modeling.

HAW Hochschule für Angewandte Wissenschaften.

IMU Inertial Measurement Unit.

LSB Least Significant Bit.

PWM Pulsweitenmodulation.

ROI Region of Interest.

ROS Robot Operating System.

SBC Single Board Computer.

SLAM Simultaneous Localization and Mapping.

uAC „Micro Autonomous Car".

uACview Telemetrie-Anwendung.

Codebeispielverzeichnis

1 Einleitung

Autonome Systeme stellen eine große Herausforderung für verschiedene Industriezweige wie Luftfahrt, Logistik und weitere dar. Speziell die Automobilindustrie ist daran interessiert, den Fahrkomfort durch Einparkassistenten und Fahrspurassistenzsysteme zu erhöhen.

Die Autopilot Technologie von Tesla übernimmt das „Lenken, Beschleunigen und Bremsen auf seiner Spur" [18], während der Fahrer das System nur noch überwachen muss. Spurwechsel hingegen müssen weiterhin vom Fahrer unternommen werden.

Die wachsende Verbreitung von IoT-Geräten und höhere Ansprüche an deren Fähigkeiten liefern immer kleiner und leistungsstärker werdende Prozessoren. Smartphones kommen herkömmlichen Computern bezüglich Rechenleistung immer näher und sind damit in der Lage, komplexe Probleme zu lösen. Gleichzeitig werden hochwertige Kameras mit kompakter Bauform für diese Geräte entwickelt. Das Startup *comma.ai* versucht zur Zeit mit seinem opensource Projekt *openpilot* [6] mit den Fahrerassistenzsystemen der Automobilindustrie zu konkurrieren. Es werden die vorhandenen Sensoren moderner Fahrzeuge verwendet und auf einem Smartphone, zusammen mit Bildern der Smartphonekamera, verarbeitet. Die Steuerung des Fahrzeugs erfolgt durch direkte Signale an die verbauten Steuergeräte.

Nicht nur auf der Straße werden aktiv autonome System erforscht. Autonom fliegende Drohnen können heute bereits Objekte verfolgen und ohne GPS durch komplexe Umgebungen fliegen [16]. Kollisionen werden durch die Kombination mehrerer Kameras verhindert. Die in Flugobjekten verwendete Hardware muss kompakt und leicht sein, damit Agilität und lange Flugzeiten realisiert werden können.

Die Verwendung von Smartphoneprozessoren für Single Board Computer (SBC) führt ebenfalls zu einer Ausweitung der Anwendungsgebiete. Sie bieten die Möglichkeit leistungsstarke, Linux-basierte Systeme mit kleinen Abmessungen zu kreieren und sind somit für kleinere Roboter gut geeignet.

Ein, in den bereits genannten Arbeitsgebieten, zum Einsatz kommendes Framework ist das Robot Operating System (ROS). Es setzt auf einem bestehenden Betriebssystem (Windows, MacOS oder Linux) auf und vereinfacht die Kommunikation zwischen verschiedenen Modulen des Robotersystems. Viele ROS-Module zur Hardwareabstraktion oder Verarbeitung von Sensoren sind frei zugänglich, wodurch leicht komplexere Systeme erstellt werden können. Häufig besitzen diese Systeme einen differentiellen Antrieb und Sensoren wie LIDAR, Drehencoder und Inertial Measurement Unit (IMU) [10].

Der von der Technischen Universität Braunschweig jährlich veranstaltete Carolo-Cup widmet sich den Herausforderungen autonomer Fahrzeuge im Maßstab 1:10 [1]. Studierende unterschiedlicher Universitäten treten in verschiedenen Disziplinen, darunter Einparken und Ausweichen von Hindernissen, gegeneinander an. Die Ansätze sind primär Kamera-basiert und verwenden weitere Sensoren zur Positionsbestimmung und Abstandsmessung zu Hindernissen. Es kommen SBCs bishin zu leistungsstarken Rechnern wie dem Intel NUC für die Auswertung zum Einsatz [3]. Oft werden Frameworks wie das zuvor genannte ROS für die Kommunikation und OpenCV für die Bildverarbeitung verwendet [4].

In einem noch kleineren Maßstab arbeitet das Miniatur Wunderland Hamburg. Die Hochschule für Angewandte Wissenschaften (HAW) arbeitet in Kooperation mit selbigem daran, ein autonomes Schiff in der miniaturisierten Welt fahren zu lassen [17]. Bisher werden die Schiffe vom Miniatur Wunderland per Hand in der H0 (1:87) Miniaturwelt gesteuert, da eine zuverlässige Automatisierung für Wasserfahrzeuge durch die Dynamik des Wassers besonders herausfordernd ist und eine robuste Umsetzung noch nicht realisiert werden konnte. Weiterhin wird seitens der HAW an einem autonom fahrenden Bus in H0-Größe gearbeitet, welcher einen RaspberryPi Zero und ein FPGA zur Beschleunigung von Aufgaben im Bereich maschinellem Lernens verwendet. Aktuell fahren die mobilen Fahrzeuge des Miniatur Wunderlandes mit Hilfe des Faller-Car-Systems und einer individuellen Datenübertragung an die Fahrzeuge [13]. Das Faller-Car-System verwendet Magneten an der Lenkung und Drähte in den Straßen, sodass die Fahrzeuge dem Draht wie Schienen folgen können. Ein zentraler Leitrechner koordiniert und steuert die Fahrzeuge.

1.1 Zielsetzung

In den folgenden Kapiteln wird die Grundlage für eine ressourcensparende, kamerabasierte, autonome Plattform gelegt. Dabei soll auch die Fahrzeugplattform, soweit möglich, minimal gehalten werden. Der Entwurf und die Realisierung von Platinen ist nicht Teil dieser Arbeit, weshalb auf fertige Platinen zurückgegriffen werden muss. Durch die Einschränkungen in Rechenleistung und Speicher von Mikrocontrollern kann nicht auf ROS und OpenCV zurückgegriffen werden. Daher muss die Grundstruktur des Systems selbst entwickelt und aufgebaut werden. Außerdem werden Softwaremodule benötigt, welche eine Telemetrie mit minimalem Ressourcenverbrauch bieten.

Durch die Teilnahme am Carolo-Cup (Basic-Cup) im Jahr 2018 mit dem TeamWorstCase konnte ein tiefer Einblick in die Herausforderungen bei der Konzeption und Umsetzung eines autonomen Systems erlangt werden. Die Beschränkung auf leichtgewichtige Algorithmen ermöglichte hohe Bildraten auf der leistungsstarken Hardware der 1:10 Fahrzeuge. Für geringere Fahrgeschwindigkeiten, wie sie in dieser Arbeit angedacht sind, ist diese Bildrate nicht nötig und somit kann Rechenleistung und damit Energie und Platz gespart werden.

Mit einem Proof-of-Concept soll auf der zuvor erstellten Basis die Disziplin *Obstacle Evasion Course* des Basic-Cups als Bewertungsmaßstab erfolgreich in einer passend zum Fahrzeug skalierten Welt abgeschlossen werden. Eingeschränkt wird diese Disziplin auf *Static Obstacles*, also stillstehende Hindernisse.

1.2 Struktur der Arbeit

Zu Beginn der Arbeit (Kapitel 2) werden die Anforderungen und die Entscheidung für die zu verwendende Hardware beschrieben. Die Hardware umfasst zum einen die elektronischen Bauteile wie Prozessor, Kamera, Leistungselektronik und Aktorik. Zum anderen behandelt das Kapitel die Wahl einer geeigneten Optik und eines geeigneten Fahrwerks. Letzteres wird nach der Analyse eines Prototypen von Grund auf neu designt.

Das Kapitel 3 enthält die Softwaregrundlagen des autonomen Systems. Anfangs werden die Anforderungen an die Softwarekomponenten sowie die Systemarchtiektur vorgestellt. Anschließend wird ein in kurzer Überblick über die Telemetrieanwendung gegeben und die Telemetrie-Protokolle für Parameter-, Bild- und Fernsteuersignalübertragung werden

beschrieben. Mit dem Einlesen von Kamerabildern wird ein weiterer Hauptbestandteil des Systems erklärt.

In Kapitel 4 werden verschiedene Ansätze zur Umsetzung von perspektivischer Bildtransformation verglichen, welche die Grundlage für die nachfolgenden Algorithmen der Bildverarbeitung darstellt. Die Algorithmen zur Fahrbahnerkennung, zur Erkennung von Hindernissen und zur Steuerung des Fahrzeugs werden anschließend beschrieben.

Das Kapitel 5 beschreibt die reale Testumgebung für das fertige Fahrzeug. Zum Testen der Algorithmen ohne Livetest wird eine minimalistische Simulation beschrieben, in der sich ein Modell des Fahrzeugs über eine Teststrecke bewegt und gleichzeitig bereits transformierte Kamerabilder simuliert werden. Anschließend werden die Eigenschaften des gesamten Fahrzeugs erläutert. Den Abschluss des Kapitels bildet ein Livetest, in dem der Wettkampf des Basic-Cups nachgestellt wird. In der Auswertung wird das Ergebnis des Livetests mit den Ergebnissen der Teilnehmer des Basic-Cups des Jahres 2018 verglichen.

Im letzten Kapitel werden die Ergebnisse der Arbeit anhand der Zielsetzung bewertet und ein Ausblick gegeben, wie dieses System erweitert werden kann.

2 Hardware

Dieses Kapitel beschreibt die Wahl der Hardwarekomponenten. Dazu gehören das Fahrwerk, die Leistungselektronik und der Prozessor. Bei der Wahl spielt ein kompakter Formfaktor eine große Rolle. Der Anlass für die Größenbeschränkungen ist die Perspektive das autonome Fahrzeug in der Skalierung 1:87 (H0) umzusetzen. Auf dem Weg zur finalen Konfiguration wurden mehrere Iterationen durchlaufen. Diese resultierten in einer Skalierung von ca 1:63.

2.1 Mikrocontroller

Die an den Mikrocontroller gestellten Anforderungen sind:

- Kompakte Bauform
- Schnelle Schnittstelle für die Bildübertragung von der Kamera
- Ausreichend Arbeitsspeicher für minimale Bildverarbeitung
- Onboard Funkverbindung (WLAN)

Der ESP32 von Espressif Systems erfüllt alle genannten Anforderungen [9]. Er ist erhältlich in der QFN48 Bauform (5mm x 5mm), wodurch kompakte Platinen möglich sind. Es steht die I2S-Schnittstelle im Parallel-Modus zur Übertragung von Bilddaten mit hoher Bandbreite zur Verfügung. Diese kann mit einem DMA-Controller angesteuert werden, wodurch weniger Rechenleistung des Prozessors für die Übertragung verwendet wird. Nach Abzug der für das Betriebssystem benötigten Speicherressourcen bleiben 300kB RAM übrig, welche für die Verarbeitung von kleineren Bildern ausreichend sind. Der ESP32 verfügt des Weiteren über ein internes WLAN- sowie Bluetooth-Modul.

Das Produkt *M5Camera* von *M5Stack* bietet ein Platinenlayout für den ESP32 inklusive 4MB Flash und 4MB PSRAM [12]. Zusätzlich ist eine passende 2MP Kamera von *Omnivision* enthalten. Durch die Ladeschaltung für Lithium-Ion-Akkus kann das Board auch kabellos mit Batterie betrieben und über ein USB-C-Kabel geladen werden. Ein besonderes Merkmal ist, dass der ESP32 bei Wechseln von USB- zu Akku-Betrieb und umgekehrt unterbrechungslos arbeiten kann. Das USB-C-Kabel wird ebenfalls zur Programmierung des ESP32s über eine serielle Schnittstelle verwendet. Der USB-Serial Adapter befindet sich auf der Platine. Im Vergleich zum verbreiteten SBC RaspberryPi Zero (65mm x 30mm) ist die M5Camera deutlich kompakter (44mm x 20mm) (Abb. 2.1).

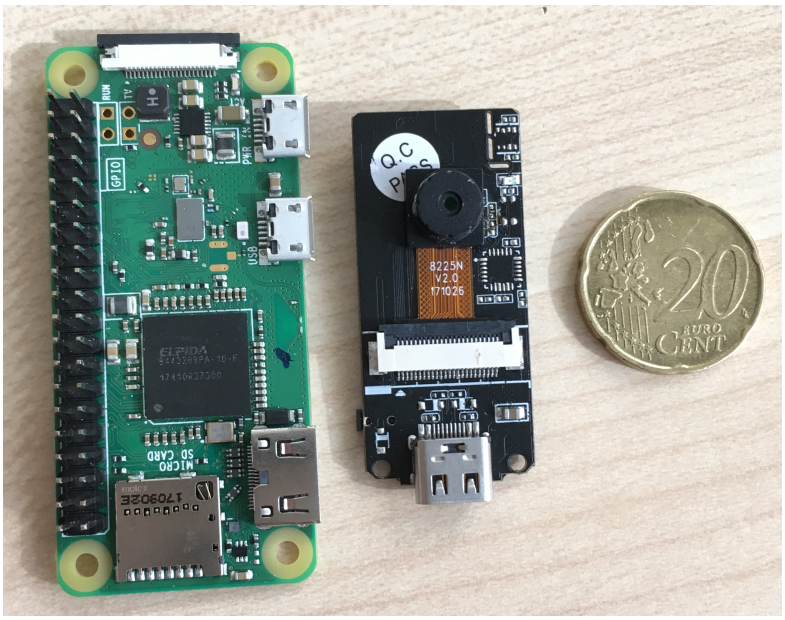


Abbildung 2.1: Vergleich RaspberryPi Zero und M5Camera

2.2 Kamera und Optik

Die verwendete Kamera OV2640 überträgt die Bilddaten über einen 8-Bit breiten Bus mit einem Pixeltakt und VSYNC- sowie HREF-Signalen [14]. Das VSYNC-Signal kündigt einen neuen Frame an. Das HREF-Signal ist hingegen solange aktiv wie valide Pixel übertragen werden. Dieses Signal kann verwendet werden, wenn nicht der ganze Frame, sondern nur ausgewählte Zeilen des Frames übertragen werden sollen. Mit jedem Pixeltakt werden 8 Datenbits parallel übertragen. Die Kamera unterstützt verschiedene Bildformate, welche byteweise über den 8 Bit Datenbus übertragen werden. In diesem

Projekt wird das Y8-Format verwendet, was einem Graubild mit Werten von 0 bis 255 entspricht.

Die im M5Camera-Paket enthaltene OV2640-Kamera besitzt eine Linse mit Blickwinkel von 78 Grad (Abb. 2.2 links). Für die Orientierung in Kurven wird ein größerer Blickwinkel benötigt um die Fahrbahn im Kurvenverlauf ausreichend sehen zu können. Daher wird ein Weitwinkel-Objektiv verwendet (Abb. 2.2 rechts), welches einen Blickwinkel von 120 Grad hat.

Vergleich von Standardobjektiv(links) und Weitwinkelobjektiv(rechts) von der gleichen Position aufgenommen.

Abbildung 2.2: Standardobjektiv vs. Weitwinkelobjektiv

Wie in Abbildung 2.3 zu sehen ist, erweitert sich das Sichtfeld der Kamera so, dass statt nur einer Fahrspur am unteren Bildrand, auf gleicher Höhe die Breite von drei Fahrspuren sichtbar wird (Beschreibung der Testumgebung in Kapitel 5.1).

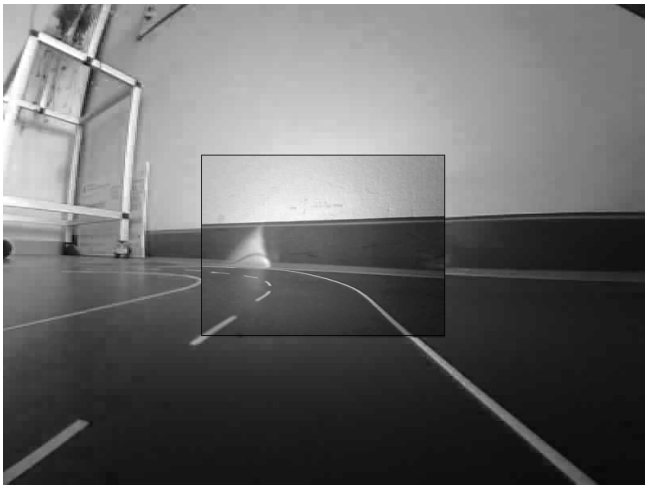


Abbildung 2.3: Standardobjektiv in Weitwinkelobjektiv

2.3 Fahrwerk

Für das minimale autonome System wird ein kompaktes Fahrwerk benötigt. Zunächst wird ein Prototyp durch Adaption eines bestehenden Modells erstellt und anschließend das finale Fahrwerk vorgestellt.

2.3.1 Adaption an bestehendes Modell

Die Recherche nach Miniaturfahrzeugen inklusive Aktorik (Antrieb und Lenkung) lieferte nur wenige nutzbare Modelle. Für erste Tests fiel die Wahl auf ein fernsteuerbares Modell der Marke *Revell* im Maßstab 1:63 [15].

Dieses Modell besitzt einen Funkempfänger, welcher die Lenkung und den Antrieb ansteuert. Die Hinterräder werden durch einen Bürstenmotor mit Untersetzung angetrieben. Die Lenkung ist über eine Magnetspule und zwei Magneten realisiert. Zwei Federn sorgen dafür, dass die Lenkung im inaktiven Zustand zur neutralen Position zurückkehrt. Abbildung 2.4 zeigt mit dicken Linien die Magnetspule. Die zwei Magneten sind mit umgekehrter Polarisierung über die Spurstange verbunden. Die gestichelten Linien stellen die zwei Federn dar. In der Magnetspule der Lenkung wird ein Feld induziert und einer der beiden Magnete auf der Spurstange wird angezogen, welches die Lenkung je nach Feldrichtung auslenkt.

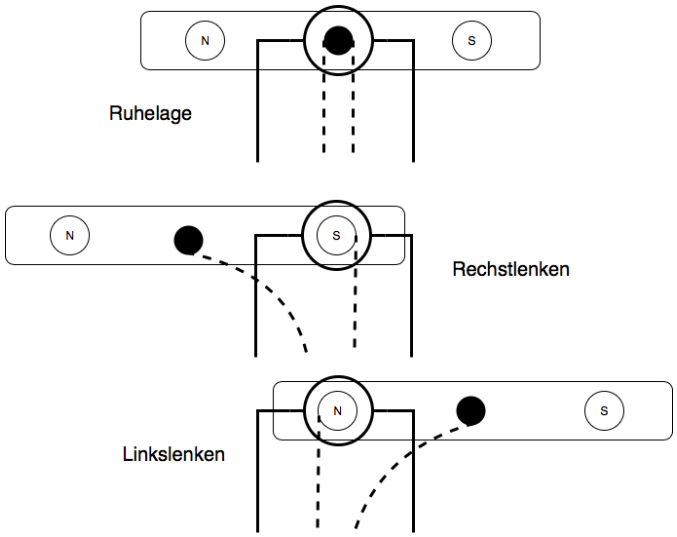


Abbildung 2.4: Magnetlenkung

Der sich bereits im Fahrzeug befindliche Funkempfänger und die Batterie werden für den Umbau entfernt. Die Lenkung und der Antrieb werden über eine Doppel-H-Brücke angetrieben. Die Ansteuerung der Leistungselektronik erfolgt über Pulsweitenmodulation (PWM).

Die Montage des Prozessors und der Kamera wird über einen 3D-gedruckten Adapter realisiert (Abb. 2.5).

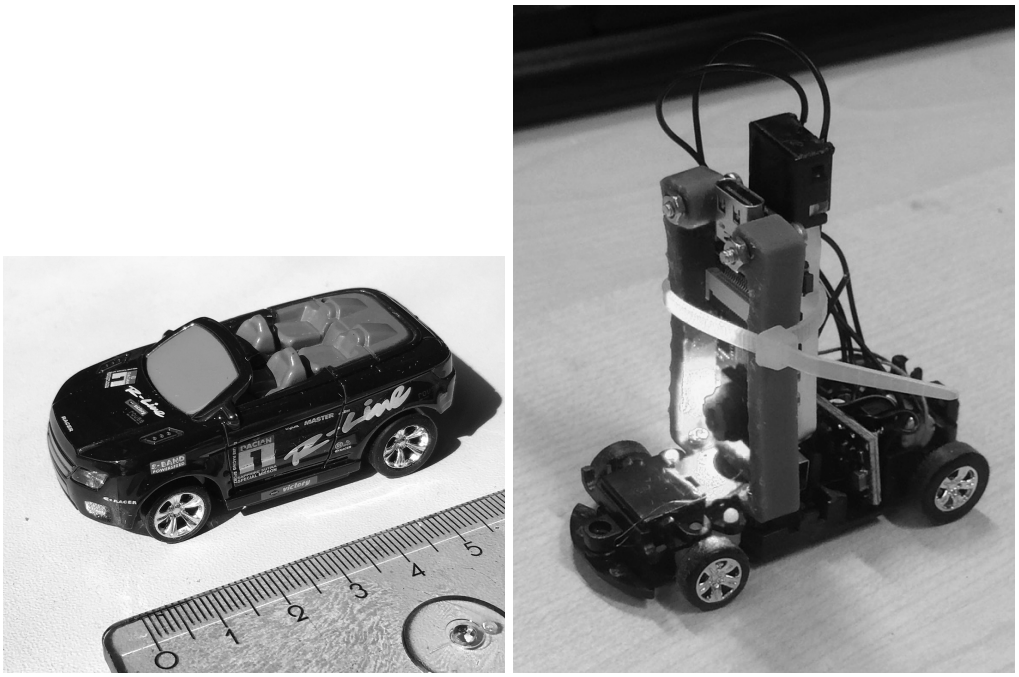


Abbildung 2.5: Umbau des gekauften Fahrwerks

Die Vorteile dieses Fahrzeugmodells sind, dass das Fahrwerk fertig produziert zu kaufen ist und mit geringem zusätzlichen Aufwand die Umwandlung zur Ansteuerung mit eigener Logik umzusetzen ist.

Das fertige Fahrzeugmodell bringt jedoch auch Nachteile mit sich: Zum Beispiel hat die Lenkung eingeschränkte Lenkwinkel (Radius 40cm), wodurch das Fahren auf proportional skalierten Fahrbahnen unmöglich wird. Die Umsetzung der Lenkung mit einer Magnetspule ist ebenfalls problematisch. Die Spule erhitzt sich bei längerer Aktivierung und kann nur drei Lenk-Positionen anfahren: Neutrallage und positive sowie negative Vollauslenkung. Kleinere Lenkwinkel sind nicht möglich.

Die Untersetzung des Antriebs ist für den Anwendungsfall des autonomen Fahrens nicht groß genug, da der Motor in den niedrigen Drehzahlen, welche für langsame Fahrten nötig sind, zu wenig Kraft besitzt, um das Fahrzeug bewegen zu können. Erst ab höheren Strömen rollt das Fahrzeug an und beschleunigt anschließend auf eine hohe Geschwindigkeit.

2.3.2 3D-gedruckte Plattform

Aus den Einschränkungen der gekauften Plattform lassen sich die Anforderungen an die finale Fahrzeugplattform ableiten:

- Lenkradien von 160mm oder kleiner und eine Ansteuerung der Lenkung, welche Teileinschläge ermöglicht.
- Die Ansteuerung von langsamen Geschwindigkeiten resultiert in einer reproduzierbaren Geschwindigkeit ohne lange Beschleunigung.
- Die Montagepunkte des Mikrocontrollers und der Kamera sind im Fahrwerk integriert.
- Die Anzahl an zu kaufenden Komponenten ist gering.
- Die Bauform der neuen Plattform soll die der gekauften Plattform nicht überschreiten.

Zur Erfüllung der Anforderungen wären viele Modifikationen der gekauften Plattform nötig gewesen. Anstatt der Modifikationen konnte mit Hilfe von Computer-Aided Design (CAD) eine neue Plattform passend zur Leistungselektronik von Grund auf neu designt und 3D-gedruckt werden.

Leistungselektronik

Die Ansteuerung der Lenkung wird über einen Linearservo realisiert. Dieser lässt sich über PWM-Signale, wie sie im Hobbybereich üblich sind, ansteuern und kann, im Gegensatz zur Magnetlenkung, auch Zwischenpositionen ansteuern. Das PWM-Signal hat eine Frequenz von 50Hz (Periode von 20000µs) und Pulsweiten von 1000µs bis 2000µs. Bei einer Pulsweite von 1500µs steht der Servo in der Mitte (Abb. 2.6).

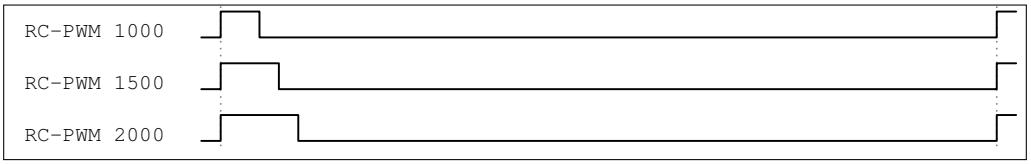


Abbildung 2.6: PWM für Linearservo

Der Antriebsstrang ist ein fertiges Produkt aus dem Modellbau für H0 Fahrzeuge. Es besteht aus einem Bürstenmotor und einem Schneckengetriebe. Über das Schneckengetriebe wird eine hohe Untersetzung erreicht, sodass langsame Geschwindigkeiten gefahren werden können. Für das Folgen einer Straßenführung ist keine Rückwärtsfahrt nötig, daher reicht eine Schaltung mit nur einem Transistor zum Antreiben des Motors aus (Abb. 2.7). Die Doppel-H-Brücke wird somit nicht mehr benötigt und kann eingespart werden. Die Geschwindigkeit kann über PWM beeinflusst werden. Anders als bei dem PWM-

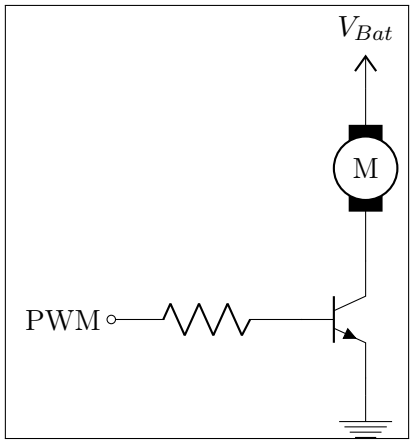


Abbildung 2.7: Motor Transistorschaltung

Signal des Servos wird für den Motor der Tastgrad, also das Verhältnis der Pulslänge zur Periode, verändert. Es wird der gesamte Bereich von 0% für Stillstand bis 100% für maximale Geschwindigkeit verwendet (Abb. 2.8).

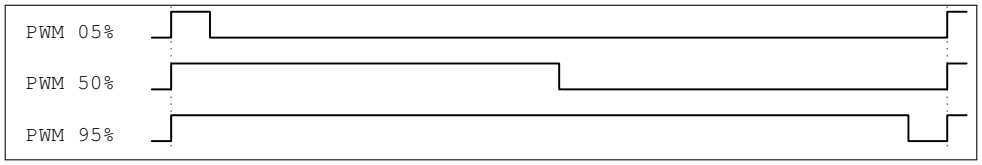


Abbildung 2.8: PWM für Motor

Fahrwerk

Der Designprozess des Fahrwerks konzentrierte sich anfangs auf die Konstruktion der Lenkung. Dafür wurden die Dimensionen der gekauften Plattform als Orientierungshilfe genutzt. Die Lenkung ist durch die beweglichen Teile der mechanisch anspruchsvollste Teil des Designs. Die benötigten zusätzlichen Komponenten Servo, Antriebsstrang, Mikrocontroller, Kamera und Akku wurden als Mockups in der Konstruktion nachmodelliert und ermöglichten eine kompakte Anordnung dieser im fertigen Design. Die Halterungen für alle Komponenten wurden anschließend konstruiert, um eine kompakte Bauweise zu erzielen.

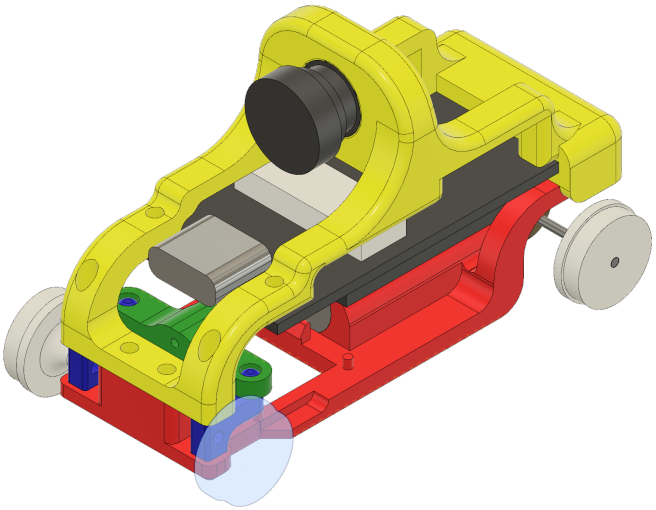


Abbildung 2.9: CAD der neuen Plattform

Das fertige Design (Abb. 2.9) umfasst fünf verschiedene zu druckende Einzelteile: Das Hauptteil ist die Bodenplatte (rot) mit Halterungen für den Antrieb, den Lenkservo sowie die Mikrocontrollerhalterung. Das zweite größere Teil des Designs ist die Mikrocontroller- und Kamerahalterung (gelb), welche über zwei Schrauben und ein Stecksystem an der Bodenplatte montiert wird. Weiterhin sind die Achsschenkel (blau), die Spurstange (grün) und die Räder (grau) enthalten.

Lenkung

Die Lenkung besteht aus den Achsschenkeln und der Spurstange, welche über den Linearservo bewegt wird. Der Servo beeinflusst nur die seitliche Auslenkung der Spurstange. Die Rotation der Achsschenkel dadurch wird erlaubt, dass der Stift, welcher Spurstange und Linearservo verbindet, im Servohorn frei gelagert ist. Der Abstand zwischen Spurstange und Servohorn vergrößert sich, je stärker die Auslenkung ist (Abb. 2.10). Die freie Lagerung des Stifts kann die Abstandsänderung kompensieren. Für die Ansteuerung des Servos ist zu beachten, dass die Auslenkung des Servos nicht proportional zum Lenkwinkel an den Achsschenkeln ist. Anstattdessen ist der Sinus des Lenkwinkels proportional zur Auslenkung des Servos (genauer in Kapitel 4.4.1).

$$sin(\alpha) \sim servo_{percent}$$

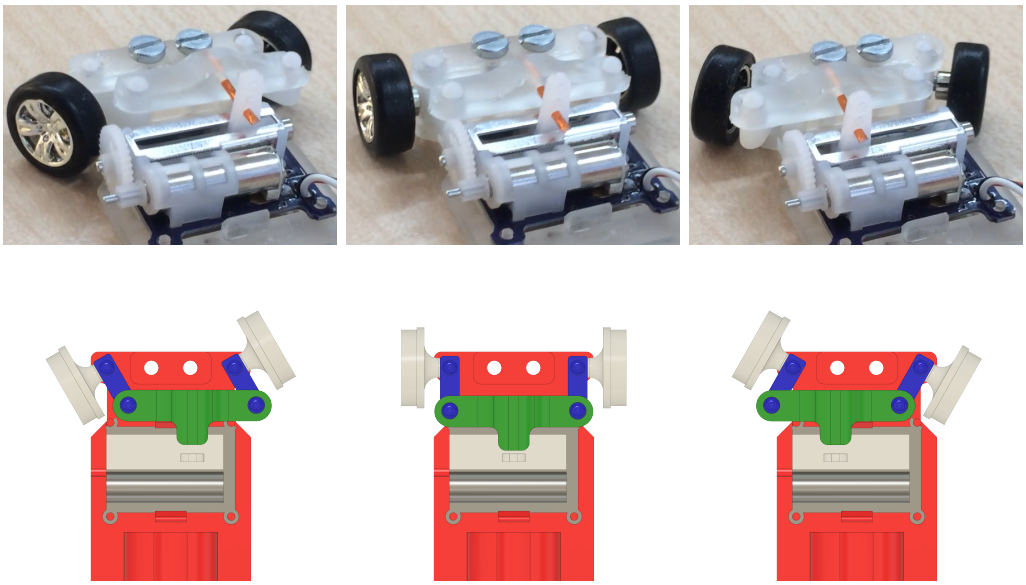


Abbildung 2.10: 3D gedruckte Lenkung mit Linearservo

Fertigung und Materialien

Zum Drucken wird der Harz-basierte 3D-Drucker *Form2* von *Formlabs* verwendet. Dieser erreicht im Vergleich zu Fused Deposition Modeling (FDM)-Druckern eine deutlich höhere Auflösung der zu druckenden Teile. Die hohe Auflösung ist nötig, da die Verbindung des

Achsschenkels zur Bodenplatte und zur Spurstange durch gedruckte Lagerungen umgesetzt ist und eine geringe Reibung an diesen Stellen für die Leichtgängigkeit der Lenkung erforderlich ist.

Um die Reibung zu minimieren sind drei verschiedene Materialien getestet worden: Die Harzvariante *Durable* ist ein bisschen flexibel und dadurch weniger brüchig. Jedoch ist die Lenkung dadurch auch schwergängiger. Die Varianten *Clear* und *Grey Pro* erzielen beide gute Resultat für die Lenkung. *Grey Pro* hat gegenüber *Clear* den Vorteil, dass Details präziser geformt sind.

Der Zusammenbau der Komponenten geschieht über zwei M2x5 Schrauben zur Verbindung zwischen Bodenplatten und Mikrocontrollerhalterung. Der Mikrocontroller wird mit zwei M2x4 Schrauben an seiner Halterung montiert. Der Antriebsstrang und der Linearservo werden in der Bodenplatte eingeklemmt. Vier Führungsstifte halten den Servo in der richtigen Position. Die Vorderräder werden mit M1.6x6 Zylinderkopfschrauben, welche als Achsen dienen, an den Achsschenkeln befestigt. Die Verbindung zwischen Spurstange und Linearservo wird über einen 1mm dicken Stift hergestellt, welcher fest in der Spurstange sitzt. Für bessere Haftung werden auf die gedruckten Räder Gummihaftreifen gezogen.

Die Kosten für die gesamte Hardware betragen 94,55 Euro ohne Arbeitszeit (Tabelle 2.1).

Teil	Preis (in Euro)	Produkt
M5Camera (FishEye)*	22,99	amazon.de B07QRQ2FPC
Optik*	10,87	amazon.de B07K5B7C83
Linearservo	12,99	conrad.de 404224 - 62
Antriebsstrang	29,99	conrad.de 247068 - 62
Haftreifen	3,89	conrad.de 214274 - 62
Akku	8,99	conrad.de 404240 - 62
Schalter	1,99	conrad.de 1377837 - 62
3D-Druckmaterial	2,84	formlabs.com *Grey Pro Resin*
Summe	94,55	

Preise vom 24.05.2019.
**Zur Zeit der Bestellung war die M5Camera noch nicht mit Weitwinkelobjektiv verfügbar, daher war eine separate Optik nötig.*

Tabelle 2.1: Stückliste

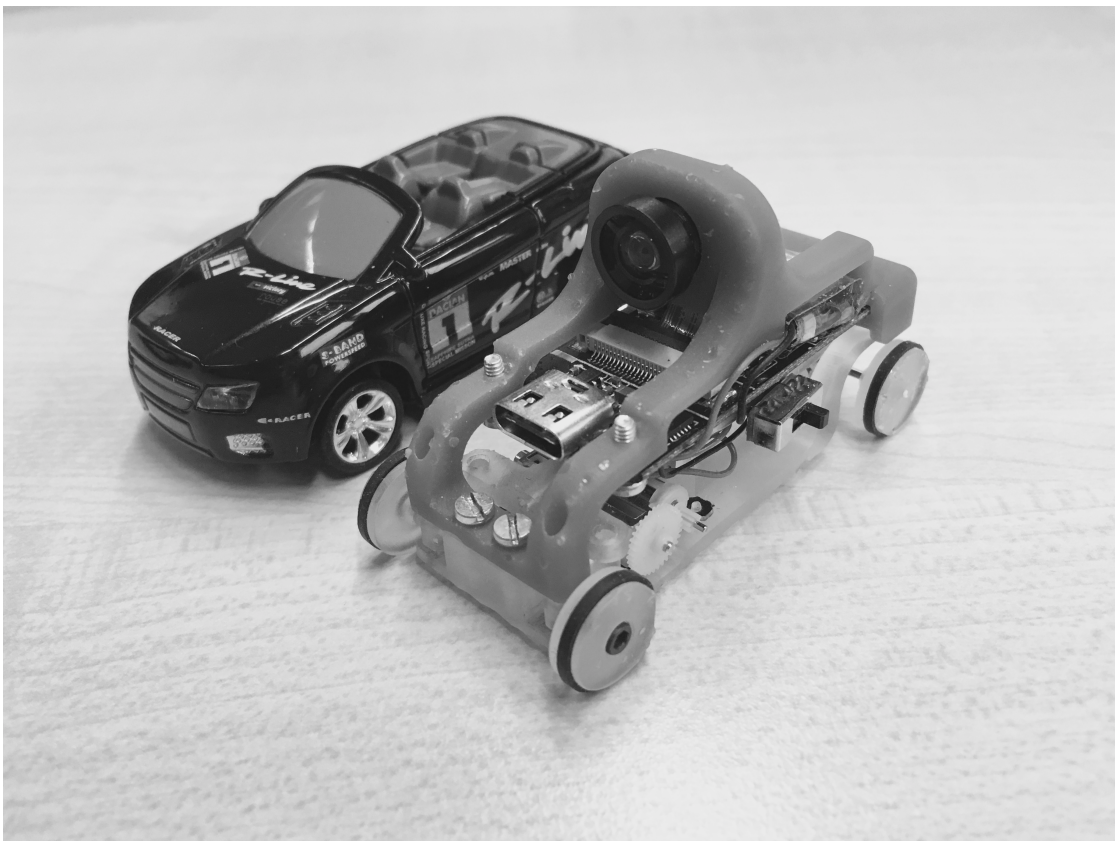


Abbildung 2.11: 3D-gedrucktes Fahrzeug vs. gekauftes Fahrzeug

3 Software

Das „Micro Autonomous Car" (uAC) ist der Name des zu entwickelnden autonomen Systems und basiert auf dem *Espressif IoT Development Framework (ESP-IDF)*. Dieses baut auf dem Echtzeitbetriebssystem *FreeRTOS* auf und bietet unter anderem einen leichten Zugriff auf das WLAN Modul und Sockets für den ESP32. Dieses Kapitel enthält die Anforderung an die Softwarekomponenten, die Systemarchitektur. Die grundlegenden Softwarekomponenten wie Telemetrie und Kamerainterface werden nachfolgend beschrieben und erklärt.

3.1 Anforderungen

Die Wahl des Mikrocontrollers führt zu zwei generellen Anforderungen an die Softwarekomponenten für das uAC. An die Telemetrieanwendung werden diese Anforderungen nicht gestellt, da diese auf leistungsstärkerer Hardware laufen kann.

SR1: Geringer Speicherverbrauch

Der limitierte Arbeitsspeicher des ESP32 erfordert sparsamen Umgang mit dieser Ressource. Wenn möglich, sind größere Strukturen, wie Bilder und Prozessabbild, so abzulegen, dass die Telemetrieschnittstellen diese ohne temporäre Puffer verarbeiten können.

SR2: Geringer Rechenaufwand

Auf dem Mikrocontroller soll Bildverarbeitung betrieben werden, was erheblichen Rechenaufwand bedeutet. Zur Verarbeitung eines Bildes steht nur begrenzt Zeit zur Verfügung bis das nächste Bild eintrifft. Daher müssen aufwändige Algorithmen optimiert

werden. Telemetriefunktionen sollen minimale Auswirkungen auf die zeitkritischen Operationen haben.

3.2 Systemarchitektur

Die Systemarchitektur des uAC (Abb. 3.1) enthält zum einen die Module Camera, Navigator, ControlSwitch und CarActuators und zum anderen die Telemetriemodule PicStreamer, DataStreamer und RemoteControl. Das Modul PicStreamer bietet eine Schnittstelle für die Telemetrieanwendung an, über die Bilder verschickt werden können. Das Modul DataStreamer bietet eine Schnittstelle für den Austausch von Daten im Binärformat an. Über das Modul RemoteControl kann die Aktorik des Fahrzeugs gesteuert werden, oder im ControlSwitch die Kontrolle an die interne Logik abgegeben werden.

Das Camera-Modul ist für die Kommunikation mit der Kamera zuständig und liefert Bilder an das nachfolgende Navigator-Modul. Im Navigator-Modul wird das Bild verarbeitet, eine Route ermittelt und die benötigten Stellwerte werden berechnet. Diese Stellwerte

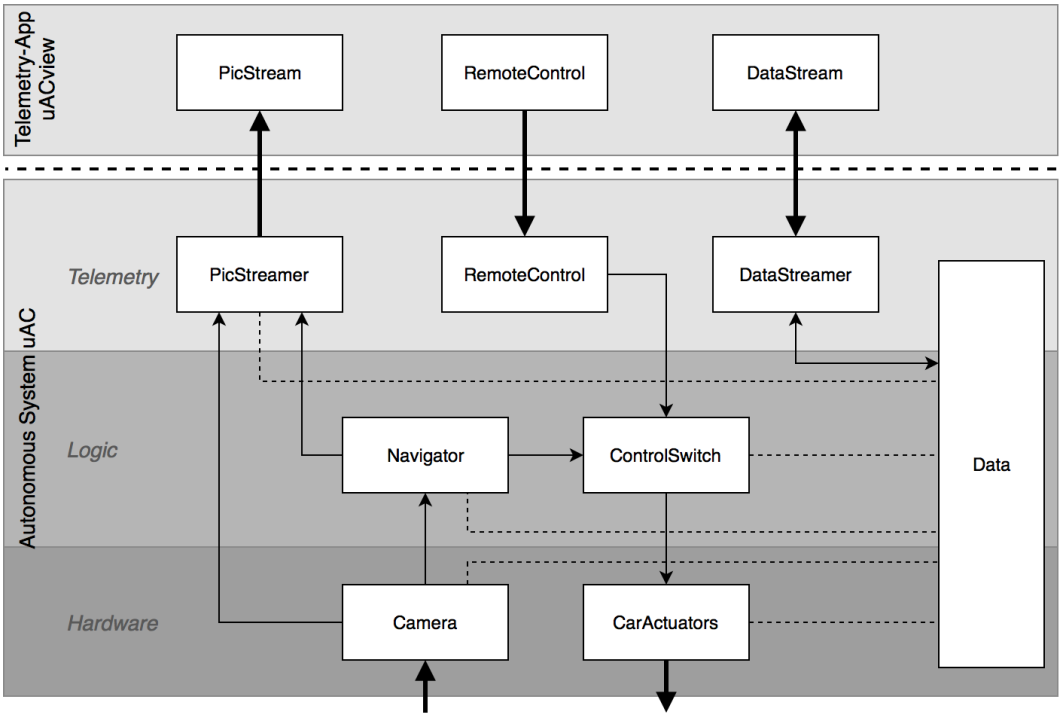


Abbildung 3.1: Schematische Darstellung der Systemarchitektur

werden dem ControlSwitch-Modul bereitgestellt. In dem ControlSwitch-Modul wird entschieden, welche Signale an die Aktoren weitergeleitet werden. Es kann zwischen den Signalen des RemoteControl-Moduls und des Navigators umschalten. Abschließend werden diese Signale von dem Modul CarActuators in die für die Hardwarekomponenten passende Form umgewandelt.

3.3 Telemetrie

In diesem Kapitel werden Besonderheiten der Konzeption und Realisierung der drahtlosen Telemetrie des uAC aufgezeigt. Die Telemetrie wird eingesetzt, um den Systemzustand des uAC abzubilden und so eine Liveanalyse seines Verhaltens zu ermöglichen.

Es werden Schnittstellen benötigt (Abb. 3.1), um interne Variablen des Systems zu lesen und zu schreiben, Bilder zu übertragen und Features in den Bildern zu markieren. Weiterhin wird eine Schnittstelle zur Fernsteuerung des uAC benötigt, mit der die Aktorik direkt angesteuert oder die Kontrolle der Aktorik an die interne Logik übergeben werden kann.

3.3.1 Telemetrie-Anwendung

Die Telemetrie-Anwendung (uACview) besitzt eine GUI zur Darstellung des Livestreams der Kamera und der Telemetriedaten des uAC. Für die Implementation der GUI wurde das QT5-Framework verwendet.

Die empfangenen Telemetriedaten werden live in einer Baumstruktur angezeigt (Abb. 3.2 unten). Die Ebenen der Hierarchie des Baums werden über Punkte getrennt. So ist die Variable `nav.arcSteering` dargestellt als Kindknoten `arcSteering` des Knotens `nav`.

Jede übertragene Variable kann live geplottet werden (Abb. 3.2 rechts oben). Dabei gibt es verschiedene Darstellungsmöglichkeiten: Die Variablen können entweder in gemeinsamen oder in separaten Plots angezeigt werden.

Für spätere Analysen werden alle empfangenen Daten im CSV-Format abgespeichert.

Der Livestream der Kamera skaliert mit der Fenstergröße (Abb. 3.2 oben links). In der dunkleren Zeile über dem Livestreambild werden Metainformationen zum Mauszeiger

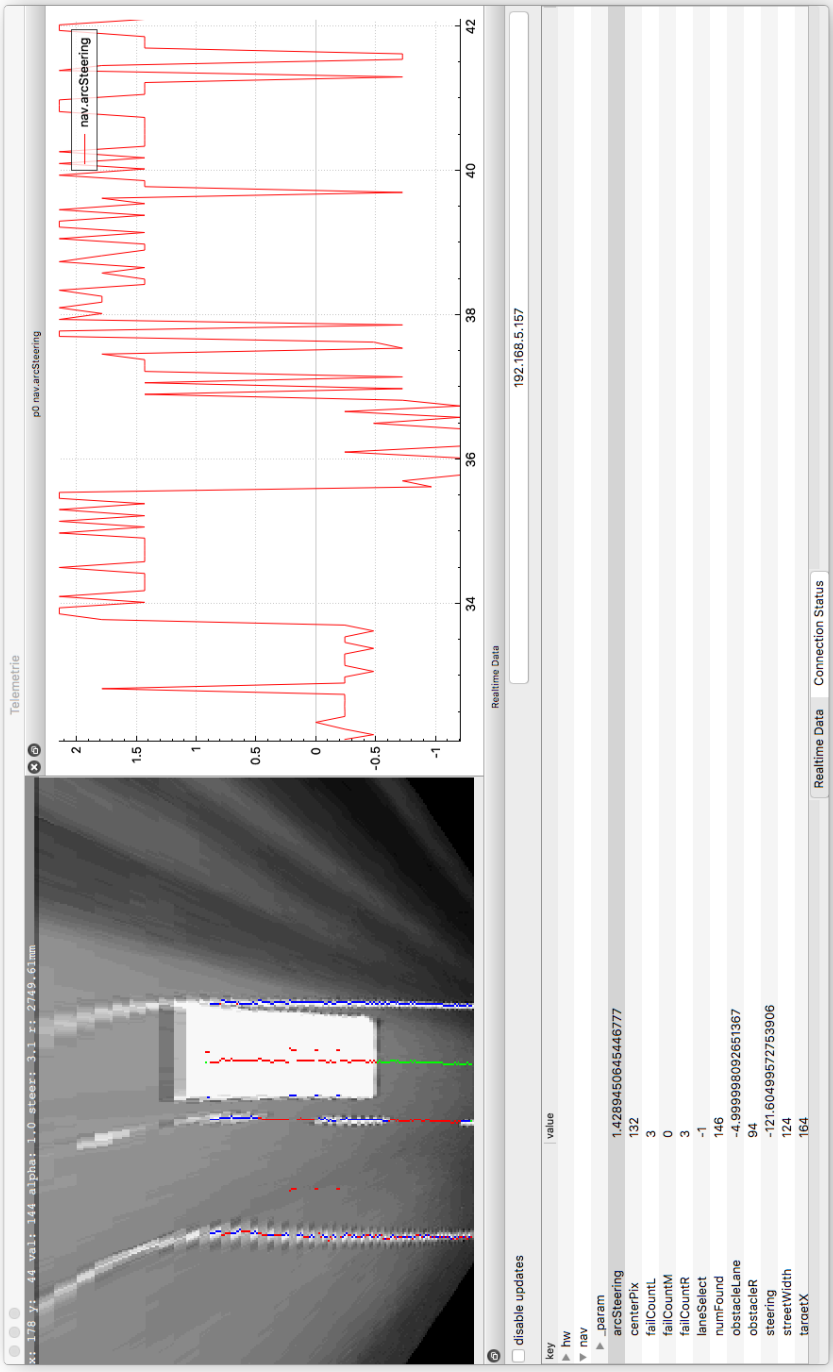


Abbildung 3.2: Telemetrie-Anwendung

dargestellt. Diese enthalten die Mauszeigerposition in Originalbildkoordinaten sowie den Farbwert an diesen Koordinaten. Weiterhin kann ein Overlay angezeigt werden, welches die speziell codierten Features in Farbe markiert (Details in Kapitel 3.3.4).

3.3.2 Daten Logging

Dieser Abschnitt beschreibt das Nachrichtenprotokoll, die Datenstruktur und das Interface für das Lesen und Schreiben von Variablen des uAC über den DataStreamer. Die Anforderungen für das Daten Logging sind folgende:

- Die uACview benötigt keine statischen Informationen über die Art der Daten. Durch das dynamische Laden dieser Informationen kann dieselbe Version der Telemetrieanwendung, unabhängig von der jeweilen Art der Daten, verwendet werden.
- Die API zum Erstellen einer geloggten Variable ist so simpel, dass der Programmierer nur minimalen Mehraufwand beim programmieren aufwenden muss.
- Die Verwendung einer Variable, welche in der Telemetrieanwendung erreichbar ist, unterscheidet sich im Code nicht von der Verwendung einer regulären Variable. Außerdem ist kein zusätzlicher Rechenaufwand, abgesehen von der Erzeugung dieser Variable, nötig.
- Verschiedene Module, welche das Daten Logging verwenden, bleiben entkoppelt und haben keine Kenntnis von den Variablen des jeweils anderen Moduls.
- Das Übertragen der Daten des uAC geschieht mit geringem Rechenaufwand (SR2) seinerseits. Das Empfangen und Verarbeiten der Daten unterliegt nicht dieser Anforderung, da hier ein leistungsstarker Rechner verwendet werden kann.

Nachrichtenprotokoll

Als Protokol zur Übertragung der Daten wird UDP verwendet. UDP ist verbindungslos und Nachrichtenorientiert. Solange keine Anfragen an das uAC gesendet wurden, werden alle ausgehenden Nachrichten an die Broadcastadresse gesendet. Ansonsten werden die Nachrichten an die Adresse der letzten eingehenden Nachricht gesendet. Durch die Verwendung von UDP wird nicht gewährleistet, dass alle Nachrichten den Empfänger erreichen. Daher müssen alle zusammengehörenden Informationen in einer einzelnen

Nachricht geschickt werden. Wenn eine Nachricht nicht übertragen werden konnte, muss diese nicht erneut übertragen werden. Dadurch stauen sich keine Nachrichten im uAC an und Anomalien durch wachsende Nachrichtenpuffer werden vermieden. Jede Nachricht besteht aus einem 32-Bit Header und den anschließenden Nutzdaten. Der Header definiert die Art der Nachricht. Es gibt vier Arten von Nachrichten:

- Die **Data**-Nachricht enthält die Werte aller Variablen. Diese werden gemeinsam in kompakter Form als Binary Large Object (BLOB) versendet. Der BLOB enthält Felder, welche jeweils einer Variable entsprechen (Tabelle 3.1). Die Größe der Felder variiert je nach Datentyp der Variable. Die Felder beinhalten die binäre Repräsentation der Variable. Es sind keine weiteren Meta-Informationen der Variable enthalten (wie zum Beispiel der Name).
- Die **Setup**-Nachricht enthält nur die Datentypen aller Variablen. Die Anzahl der Variablen entspricht der Länge der Nachricht ohne den Header.
- Die **Name**-Nachricht besteht aus dem 32-Bit Index der Variable und dem Namen der Variable als String.
- Die **Update**-Nachricht beginnt ebenfalls mit dem 32-Bit Index der zu aktualisierenden Variable. Darauf folgt die binäre Repräsentation des neuen Werts der Variable.

0x00	Header (4 Byte)	
0x04	int16_t	uint16_t
0x08	int32_t	
0x0c	...	

Tabelle 3.1: Beispiel Daten-Nachricht: Header und BLOB

Datenstruktur

Die Klasse `Data` (Abb. 3.3) kapselt alle Variablen, welche mit der Telemetrieanwendung ausgetauscht werden. Der zuvor beschriebene BLOB und Header sind im Array `assembledMsg` gespeichert (SR1). Das Array `entries` enthält die Struktur des BLOBs. Jedes Element von `entries` besitzt einen Zeiger auf den Namen der Variable, die Position im BLOB und den codierten Typen (siehe Unterkapitel Typ-Bestimmung). Weiterhin werden in `Data` die Anzahl der registrierten Variablen `numEntries` und die Position des ersten freien Feldes im BLOB gespeichert.

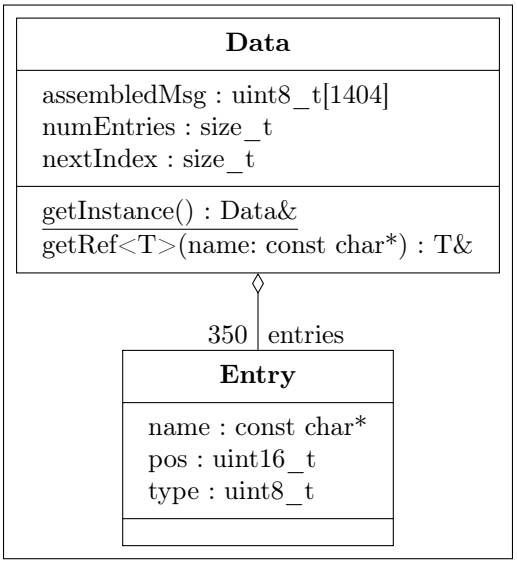


Abbildung 3.3: Klassendiagramm Data

Die Methode `getRef` von `Data` gibt die Referenz auf die Variable mit angegebenem Namen zurück (Codebeispiel 3.1). Der Typ der Variable wird als Templateparameter an die Methode übergeben. Die erhaltene Referenz zeigt direkt auf die binäre Repräsentation im BLOB.

```
1  //Signatur
2  struct Data {
3      ...
4      template<typename T>
5      T& getRef(const char* name);
6      ...
7  } data;
8
9  //Erzeugen und Initialisieren einer Variable
10 int& exampleVar = data.getRef<int>("exampleVar") = 1;
11 //Erzeugen einer zweiten Referenz auf dieselbe Variable
12 int& sameVar = data.getRef<int>("exampleVar");
```

Codebeispiel 3.1: Variable Anlegen

Falls die Variable vor dem Aufruf von `getRef` noch nicht existierte, wird diese automatisch erzeugt. Beim Erzeugen der neuen Variable wird ein neuer Eintrag für die Variable in `entries` eingefügt. Die Variable wird beim Erzeugen nicht initialisiert.

Zu beachten ist, dass es sich um ein Multithreading-System handelt. Daher ist die Funk-

tion `getRef` thread-safe implementiert. Die Verwendung der von `getRef` zurückgegebenen Variable (`exampleVar`) ist hingegen nicht thread-safe (Codebeispiel 3.1 Zeile 10). Hier ist der Programmierer, falls nötig, verantwortlich für eine entsprechende Implementation der Synchronisation.

Typ-Bestimmung

Die Codierung der Datentypen der Variablen ist beschränkt auf atomare Typen und erfolgt durch template Programmierung und die Verwendung von type_traits. Der Typ wird in einem Byte codiert (Codebeispiel 3.2). Zuerst wird die Größe des Datentyps in Bytes ermittelt und in Bit-0 bis Bit-3 geschrieben (Zeile 5). Anschließend wird überprüft, ob es sich um eine Fließkommazahl handelt (Zeile 6) und bei Bedarf das Bit-7 gesetzt.

Handelt es sich um einen Integer-Typen, wird ermittelt, ob es ein vorzeichenbehafteter Typ ist (Zeile 9). Bei vorzeichenbehafteten Typen wird das Bit-6 gesetzt.

Die resultierenden Type-Codes können aus Tabelle 3.2 entnommen werden.

```
1  template<typename T>
2  constexpr uint8_t toType(){
3      const auto size = sizeof(T);
4      static_assert(size < 16, "could not encode size -> needs new feature");
5      uint8_t ret = size;
6      if (std::is_floating_point<T>::value){
7          ret |= 0x80; //set float flag
8      } else {
9          if(std::is_signed<T>::value){
10             ret |= 0x40; // set signed flag
11         }
12     }
13     return ret;
14 }
```

Codebeispiel 3.2: Typ-Kodierung

Typ	Code
uint8_t	0x01
uint16_t	0x02
⋮	
int32_t	0x44
int64_t	0x48
float(32Bit)	0x84
double(64Bit)	0x88

Tabelle 3.2: Typ-Codierung

Serialisierung und Deserialisierung

Die Variablen werden in ihrer binären Repräsentation übertragen. Damit entfällt auf der Seite des uAC der Aufwand der Serialisierung und Deserialisierung. Dieser Aufwand wird dafür auf der Seite der Telemetrie-Anwendung betrieben. Die Factory bekommt die Typen aus der Setup-Nachricht. Für jedes Feld wird ein Feld-Objekt erzeugt. Diese Feld-Objekte enthalten einen Namen, den Typen des Feldes und zwei Funktionen:

- `bin2readable`: Deserialisierung der binären Darstellung zu einer lesbaren Darstellung (String)
- `readable2bin`: Serialisierung der lesbaren Darstellung (String) zu der entsprechenden binären Darstellung dieses Datentyps

Erhält die Telemetrie-Anwendung eine Data-Nachricht, so wird auf dieser Nachricht von jedem Feld-Objekt `bin2readable` aufgerufen. Auf diese Weise extrahiert jedes Feld-Objekt die ihm zugehörigen Daten und kreiert eine lesbare Repräsentation, welche in der GUI dargestellt werden kann.

Wird in der GUI ein Feld verändert, so wird von dem assoziierten Feld-Objekt `readable2bin` aufgerufen und eine Update-Nachricht mit der binären Darstellung des Feldes verschickt.

Die tatsächliche Umwandlung zwischen lesbarer und binärer Repräsentation wird in QT durch `QDataStream` mit Streamoperatoren und `QVariant` realisiert. Für eine Pythonvariante kann die Bibliothek `structs` effektiv verwendet werden (Beispielprogramm im digitalen Anhang).

Ablauf

Für eine korrekte Interpretation der BLOBs muss die Telemetrie-Anwendung die Struktur des BLOBs erfragen: Zuerst werden die Datentypen der Variablen angefordert (Setup-Nachricht). Anschließend wird für jede Variable einzeln der Name abgefragt (Name-Nachricht) (Abb. 3.4). Nachdem die Struktur vollständig übertragen wurde verschickt das uAC seinen Systemzustand nach der vollständigen Verarbeitung eines jeden Frames (Data-Nachricht).

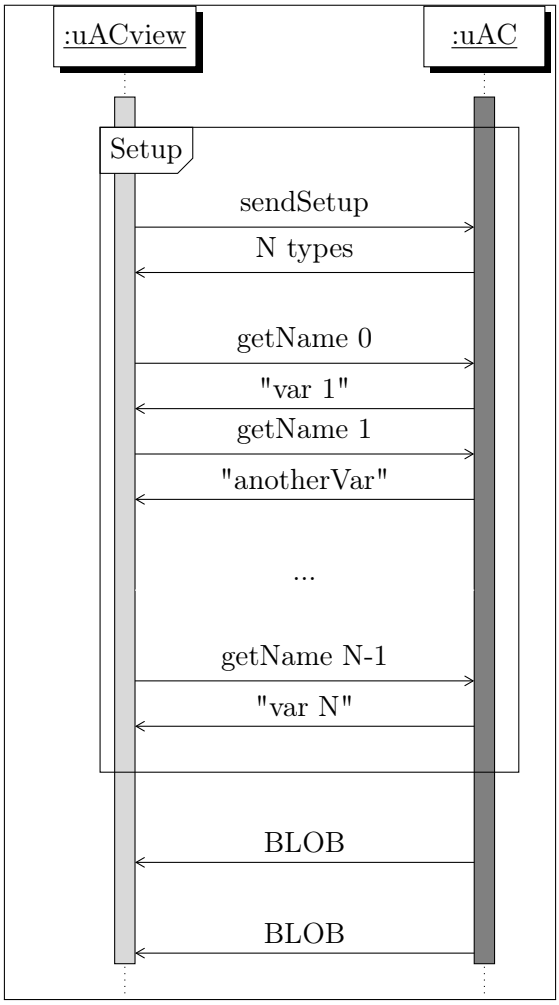


Abbildung 3.4: Sequenzdiagramm Telemetriedatenaustausch

3.3.3 Fernsteuerung

Die Fernsteuerung ermöglicht die komplette Steuerung der Aktorik. Weiterhin verhindert sie ungewollte Aktionen des Fahrzeugs, da die Kontrolle der Aktorik durch die interne Logik ebenfalls durch die Fernsteuerung aktiviert wird.

Die Fernsteuerungssignale werden über UDP übertragen. Es gibt zwei Typen von Nachrichten. Der erste Typ besteht aus nur einem Byte, welches Flags codiert (Tabelle 3.3). Der zweite Typ enthält jeweils einen Floatingpoint-Wert für Lenkung (-100% bis 100%) und Geschwindigkeit (0% bis 100%). Der Nachrichten-Typ kann frei gewählt werden.

Bit	Flag
0	linkslenken
1	rechtslenken
2	vorwärts
3	selbstständiges Fahren
4...6	*reserviert*
7	schnell

Tabelle 3.3: Fernsteuerungs-Flags

Die Aktorik des uAC ist über den ControlSwitch deaktiviert, solange keine Fernsteuerungssignale empfangen werden (Abb. 3.5). Ein Watchdog sorgt für die Deaktivierung der Aktorik, wenn mehr als eine Sekunde seit dem letzten Empfang eines Fernsteuerungssignals vergangen ist. Dadurch kehrt das uAC nach spätestens einer Sekunde in den inaktiven Zustand zurück, wenn die Verbindung beispielsweise durch Ausfall der Funkverbindung unterbrochen wird.

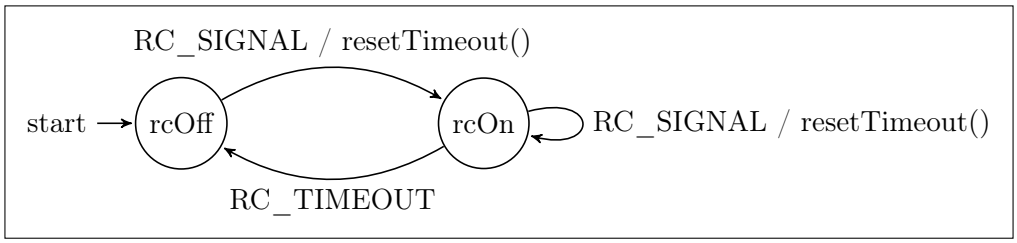


Abbildung 3.5: Zustandsautomat Fernsteuerung

Um den Abbruch der Verbindung zu verhindern und eine reaktive Steuerung des Fahrzeugs zu gewährleisten werden die Steuersignale von der Telemetrieanwendung mit einer Frequenz von 50Hz gesendet.

3.3.4 Bildübertragung

Die Bildübertragung (PicStreamer) an die Telemetrieanwendung wird dazu benötigt, die Bilder an verschiedenen Stellen der Bildverarbeitung sehen zu können.

Protokoll

Die zu übertragenden Bilder können an verschiedenen Stellen in der Bildverarbeitung verschiedene Größen haben. Daher ist das Protokoll so gewählt, dass Bilder verschiedener Größen und Formate unterstützt werden. Ohne Kompression muss jeder Pixel des Bildes und ein Header, welcher das Bild beschreibt, übertragen werden. Ein Frame der Auflösung 800x100 Pixel (Kapitel 4.1) in 8 Bit Graustufen benötigt demnach mehr als 80.000 Bytes. Diese Größe überschreitet die maximalen Größen von Ethernetframes und UDP-Paketen in der ESP-IDF. Deshalb wird die Bildübertragung über TCP realisiert, um eine eigene Auftrennung in mehrere Nachrichten und deren Zusammenführung in der Telemetrieanwendung zu vermeiden.

Frame

Bilder werden als Frames gespeichert. Frames entsprechen einem Pixelbuffer und einem Header, so dass sie dem Protokoll der Bildübertragung genügen (SR1). Der Header hat zwei Felder (Tabelle 3.4). Das erste Feld beschreibt die Framegröße in Bytes. Das zweite Feld gibt die Bildbreite in Pixeln an. Die Bildhöhe kann durch eine Division von Framegröße durch Bildbreite ermittelt werden. Ein Sonderfall ist die Bildbreite von 0, wodurch signalisiert wird, dass ein Bild codiert als Datei gesendet wird (z.B. JPEG).

Header		Payload
Framegröße	Bildbreite	Pixeldaten...

Tabelle 3.4: Framestruktur

Außerdem bietet der Frame eine Abstraktion vom reinen Pixelbuffer (Codebeispiel 3.3). Der Zugriff auf Pixel mit X- und Y-Koordinaten wird angeboten und Überschreitungen der Bildgrenzen können abgefangen werden.

```
1 Frame input(800, 100);
2
3 //use of operator(int x, int y)
4 auto& validPixel = input(10, 10);
5 validPixel = 255;
6
7 //use of operator(int x, int y)
8 auto& invalidPixel = input(-10, 10);
9 invalidPixel = 255;
```

Codebeispiel 3.3: Pixelbufferabstraktion

In Zeile 8 des Codebeispiels 3.3 wird die Referenz eines Pixel an den Koordinaten (-10, 10) des Bildes abgerufen. Diese Koordinaten liegen außerhalb des Bildes und müssen besonders behandelt werden. Durch den Zugriff auf den Pixel über den Klammer-Operator wird geprüft, ob die angegebenen Koordinaten innerhalb des Bildbereichs liegen. Wenn die Koordinaten außerhalb liegen, wird ein extra Pixel zurückgegeben, welcher immer den Wert 0 enthält. Dieser Pixel darf wie jeder andere Pixel innerhalb des Bildes verändert werden (Zeile 9), wobei die Änderungen nicht bestehen bleiben. Beim nächsten Zugriff auf einen Pixel außerhalb des Bildbereichs über den Klammer-Operator wird der extra Pixel wieder auf den Wert 0 gesetzt.

Diese Variante zur Behandlung von Grenzüberschritten verringert die Komplexität des Codes, wenn auf Frames zugegriffen werden muss, denn Grenzen müssen nicht an allen Stellen im Code einzeln geprüft werden. Ein Nachteil ist, dass nicht unterschieden werden kann, ob der erhaltene Pixel ein extra Pixel oder ein normaler Pixel des Bildes ist. Ohne diese Information kann kein Abbruch des Algorithmus erfolgen, sobald er die Grenzen des Frames verlässt.

Einschränkungen

Aufgrund des limiterten Arbeitsspeichers des ESP32 kann kein extra Puffer für die Bilder, welche übertragen werden sollen, verwendet werden. Deshalb kann zum Beispiel die (Lock freie) asynchrone Drei-Buffer Kommunikation von Reto Carrara [5] nicht verwendet werden, um zu gewährleisten, dass ein übertragener Frame konsistente Daten aus nur einer Aufnahme enthält. In den Speicher passen nur zwei Frames: Das Bild der Kamera wird in einen Frame eingelesen und ein weiterer Frame wird für die Transformation

(Kapitel 4.1) und die Algorithmen verwendet. Deshalb greift der PicStreamer direkt auf einen der beiden Frames zu, wenn er ihn überträgt. Es ist daher möglich, dass sich die Daten des Frames während seiner Übertragung ändern. Besonders bei höheren Framerate ist dies zu erwarten.

Feature-Steganographie

Die Übertragung von erkannten Features in Bildern kann durch eine Codierung im Least Significant Bit (LSB) realisiert werden. Hierdurch reduziert sich die Farbtiefe um ein Bit von 8 auf 7 Bit. Im LSB kann so zum Beispiel eine erkannte Linie oder ein Hindernis markiert werden. Das Bild bleibt dadurch für das menschliche Auge weitgehend unverändert, weil das Bit mit der niedrigsten Wertigkeit kaum Einfluss auf die wahrgenommene Helligkeit hat(Abb. 3.6). Durch einen Filter können diese Features jedoch gefärbt und erkennbar gemacht werden (Abb. 3.7).

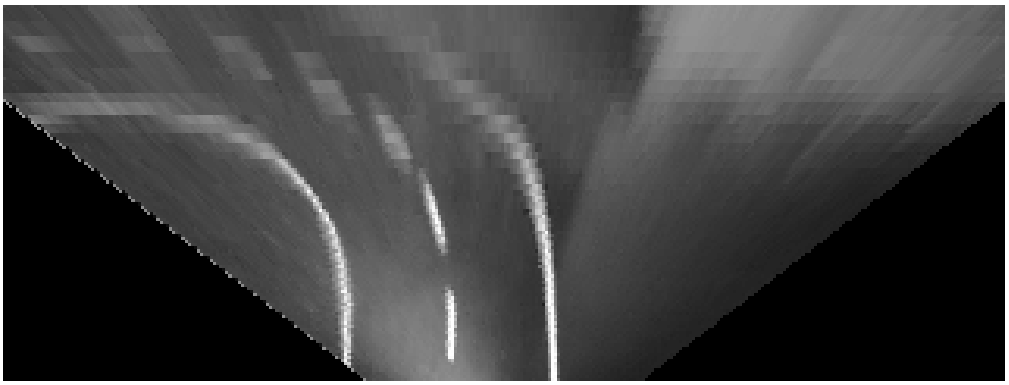


Abbildung 3.6: Feature Steganographie ohne Filter

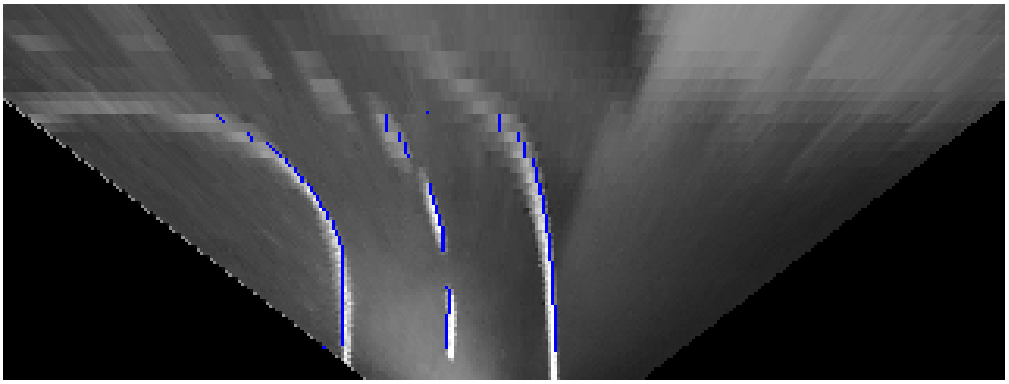


Abbildung 3.7: Feature Steganographie mit Overlayfilter

Diese Kodierung ist leicht und schnell durch ein bitweises UND mit einer Maske (0xFE) umzusetzen. Diese Operation kann direkt ausgeführt werden während der Frame transformiert wird (Kapitel 4.1). Features werden anschließend in entsprechenden Pixeln durch bitweises ODER mit 0x01 markiert.

Die Feature-Steganographie stellt sicher, dass erkannte Features einem Frame zugeordnet werden können, da sie direkt im Frame hinterlegt sind. Es sind keine weiteren Datenstrukturen zur Übertragung der Features nötig.

3.4 Kamera

Die Implementation des Kamerainterfaces orientiert sich an dem mitgelieferten Beispielcode von *M5Stack* [11]. Die Kamera lässt sich über ein I2C-ähnliches Protokoll konfigurieren. Über dieses Protokoll werden zum Beispiel die Auflösung und Region of Interest (ROI) eingestellt.

Die Übertragung der Bilddaten eines Frames wird von der Kamera gesteuert. Die Informationen werden Pixel für Pixel übertragen. Die Kamera liefert einen Pixeltakt, welcher parallel acht Bit übertragt. Zusätzlich zeigt das VSYNC-Signal, dass ein neuer Frame beginnt und das HREF-Signal, wann der Pixeltakt Daten innerhalb der ROI überträgt. Alle Takte bei inaktivem HREF-Signal können daher ignoriert werden, denn diese Pixel liegen außerhalb des ROI.

Der ESP32 kann mit dem I2S-Modul im Kameramodus die von der Kamera ausgehenden Daten einlesen [8, S. 314]. Die Daten des I2S-Moduls werden über einen Direct Memory Access (DMA)-Controller in einen Zeilenbuffer geschrieben. Sobald eine Bildzeile eingelesen wurde, wird ein Interrupt ausgelöst und die Bildzeile wird in den resultierenden Frame kopiert. Hierbei ist zu beachten, dass das I2S-Modul immer zwei Pixelinformationen (2 x 8 Bit) in 32 Bit ablegt. Deshalb kann die Bildzeile nicht direkt kopiert werden, sondern es muss explizit über alle Pixel iteriert werden.

Nach der letzten Bildzeile wird über einen Interrupt auf dem VSYNC-Signal überprüft, ob die Anzahl der übertragenen Zeilen der Erwartung entspricht. Anschließend wird ein Semaphor freigegeben, sodass die nachfolgende Bildverarbeitung angestoßen wird. Wurde nicht die richtige Anzahl an Bildzeilen übertragen, wird dieser Frame verworfen und die nachfolgende Bildverarbeitung übersprungen.

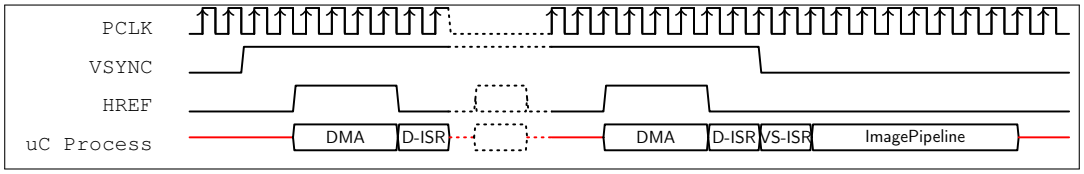


Abbildung 3.8: Kamerasignale

Den Ablauf des Einlesens kann man in Abbildung 3.8 ablesen: Zu sehen ist auf der linken Seite der Beginn eines neuen Frames mit der steigenden Flanke des VSYNC-Signals. Es folgt der Beginn der ersten Bildzeile innerhalb der ROI mit der steigenden Flanke des HREF-Signals. Die folgenden Takte der Pixelclock lösen im DMA-Treiber die Übertragung der Pixelinformationen aus. Die fallende Flange des HREF-Signals signalisiert das Ende der Bildzeile. Die D-ISR (DMA-Interrupt) wird aktiv und kopiert die Pixelinformationen aus dem DMA-Buffer an die richtige Stelle im Frame. Weitere Bildzeilen werden eingelesen, bis das Ende des neuen Frames durch die fallende Flanke des VSYNC-Signals die VS-ISR (VSYNC-Interrupt) auslöst. Anschließend wird die Bildverarbeitung des komplett eingelesenen Frames angestoßen.

4 Algorithmen

Dieses Kapitel beschreibt die verschiedenen Teile der Logik (Navigator) des uAC, welche auf den Grundlagen des vorigen Kapitels aufbauen. Die perspektivische Projektion des Kamerabildes ist die Grundlage für die Fahrbahnerkennung. Die Transformation ist aufwändig und daher werden verschiedene Optimierungen des Algorithmus verglichen. Anschließend werden die Algorithmen zur Fahrbahnerkennung und zur Vermeidung von statischen Hindernissen erläutert. Abschließend werden zwei verschiedene Ansätze zur Lenkungsregelung verglichen.

4.1 Transformation

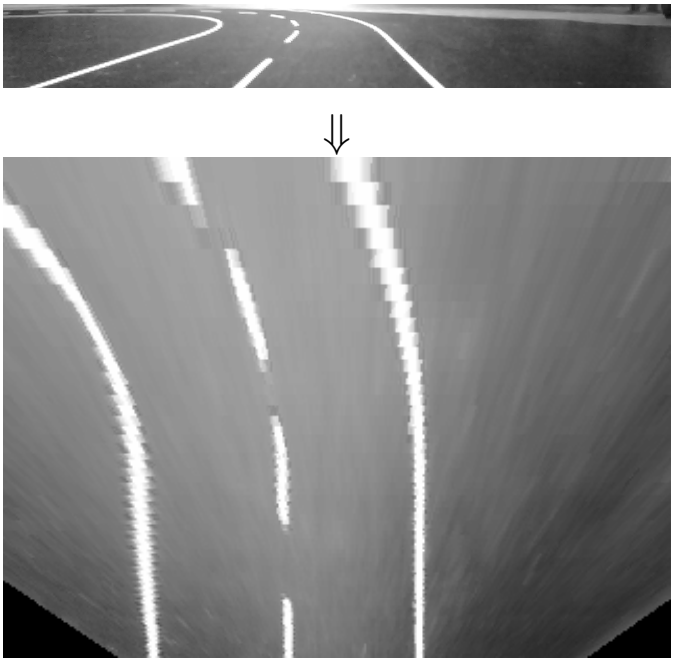


Abbildung 4.1: Transformation von Kamera- zu Vogelperspektive

Die Transformation der Frames in die Vogelperspektive (Abb. 4.1) vereinfacht die nachfolgenden Schritte der Bildverarbeitung. Sie wird mittels folgender Formel berechnet:

$$dst(x, y) = src\left(\frac{M_{11}x + M_{12}y + M_{13}}{M_{31}x + M_{32}y + M_{33}}, \frac{M_{21}x + M_{22}y + M_{23}}{M_{31}x + M_{32}y + M_{33}}\right)$$

Die 3x3-Matrix M ist konstant, solange sich Kameraposition, Blickrichtung, Optik und Auflösung nicht ändern. Für das Testfahrzeug wurde folgende Matrix ermittelt (siehe Kapitel 4.1.1):

$$\begin{pmatrix} 0.3369230769230668 & -0.4982787274453819 & 100.7753561253538 \\ 0 & 0.01186120014244981 & 5.839610042734924 \\ 0 & -0.001245696818613456 & 0.3867076210826123 \end{pmatrix}$$

Diese Matrix transformiert das Urpsrungsbild mit der Auflösung 800 x 100 Pixel in des Zielbild mit der Auflösung 320 x 240 Pixel. Im Zielbild entspricht ein Pixel einem Quadratmillimeter (1mm x 1mm). Abbildung 4.2 zeigt den Sichtbereich des Fahrzeugs nach der Transformation.

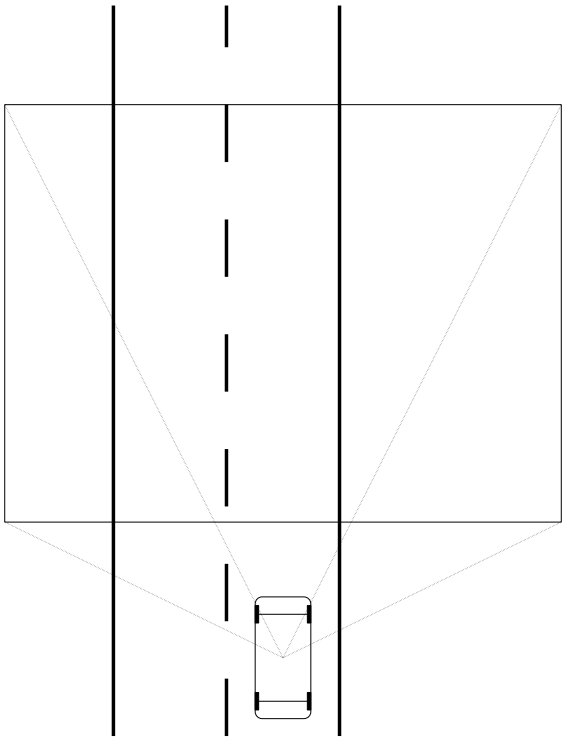


Abbildung 4.2: Sichtbereich des Fahrzeugs

Die Transformation wird auf jeden Frame angewendet und trägt einen wesentlichen Teil zum Rechenaufwand bei. Verschiedene Ansätze der Implementation werden nachfolgend auf Performanz verglichen:

Algorithmus	Zeit
Referenz Iteration	6ms
On-demand Float	156ms
On-demand Float ohne Multiplikation	140ms
Look-up-Table	48ms
On-demand Fixpoint ohne Multiplikation	16ms

Tabelle 4.1: Rechenzeit Transformationsimplementationen

Zur Messung der benötigten Zeit wird ein GPIO am Mikrocontroller zu Beginn und zum Ende der Berechnung geändert. Diese Signale wurden mit einem Logic Analyzer gemessen. Als Referenzzeit dient die Iteration über das gesamte Zielbild, sodass jeder Pixel gelesen, modifiziert (bitweises Und) und geschrieben wird.

On-demand Float

Der langsamste Ansatz für die Transformation ist die direkte Umsetzung der Berechnung mit 32-Bit Fließkommazahlen (Tabelle 4.1). Selbst die Optimierung des Algorithmus durch Ersetzen von Multiplikationen durch Additionen reduziert die benötigte Zeit nur um 10%. Dies ist möglich, da durch alle Koordinaten in ganzzahligen Schritten iteriert wird.

```
1  // Mit Multiplikation (Vereinfacht)
2  for (int col = 0; col < WIDTH; ++col){
3      float x_src = col * M_11;
4      //benutze x_src fuer die Transformation
5  }
6  // Mit Addition
7  float x_src_tmp = 0;
8  for (int col = 0; col < WIDTH; ++col){
9      float x_src = x_src_tmp;
10     x_src_tmp += M_11;
11     //benutze x_src fuer die Transformation
12 }
```

Codebeispiel 4.1: Optimierung: Vermeidung von Multiplikation

Das vereinfachte Codebeispiel 4.1 zeigt, wie die Multiplikation der Laufvariable `col` und `M_11` durch eine Addition ersetzt werden kann, wenn das Ergebnis der vorherigen Berechnung in der nachfolgenden Iteration noch bekannt ist.

Look-up-Table

Eine deutliche Zeitersparnis wird durch Vorberechnung der Ursprungskoordinaten erreicht. Die vorberechneten Koordinaten werden in einer großen Look-up-Table gespeichert. Die Target-to-Source Transformation wird in der Look-up-Tabelle direkt über Indizes repräsentiert und kann so simpel und schnell ausgeführt werden (Codebeispiel 4.2).

```
1 const int lut[numPixels] = {/* indizes ... */};
2 for(size_t i = 0; i < numPixels; i++){
3     dst[i] = src[lut[i]];
4 }
```

Codebeispiel 4.2: Bildtransformation mit Look-Up-Table

Die Geschwindigkeit dieser Implementation wird hauptsächlich durch das Laden aus dem Flash limitert, denn die Tabelle muss auf Grund der ihrer Größe im Flashspeicher abgelegt werden. Bei einer Framerate von 12.5fps bleiben für die Verarbeitung eines Frames nur 80ms, weshalb die Transformation mit 48ms schon einen Großteil (60%) der verfügbaren Zeit verbraucht. Zu beachten ist, dass bei der Implementation mit Look-up-Table die Transformation beliebig komplex sein kann (z.B. gleichzeitig eine Linsenkorrektur und perspektivische Projektion) während die benötigte Zeit sich nicht verändert. Dies ist für Systeme mit größerem Speicher gut geeignet.

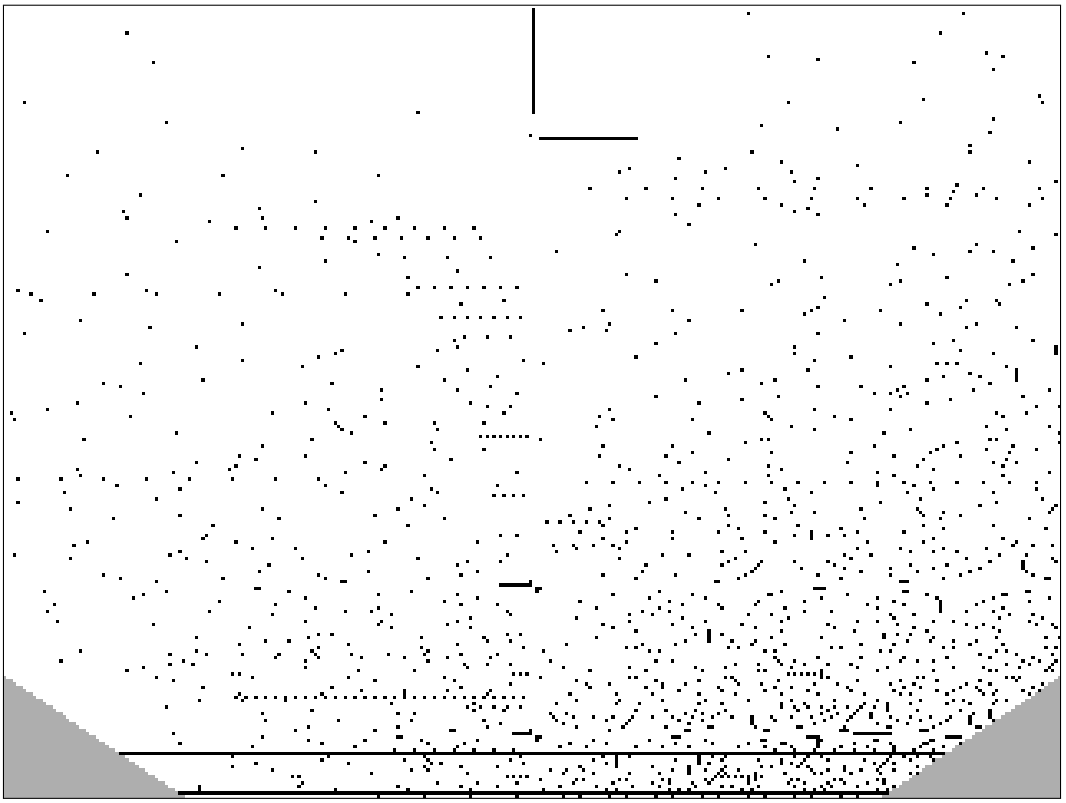
On-demand Fixpoint

Die Berechnung der Ursprungskoordinaten mit Festkommazahlen ist schneller als alle zuvor beschriebenen Verfahren und benötigt lediglich 16ms. So bleiben noch 64ms für die weitere Verarbeitung. Die Matrix M wird mit dem größtmöglichen Faktor a multipliziert, sodass der Wertebereich eines 32 Bit Integers für keinen Pixel verlassen wird. Der Faktor a ist abhängig von der auszuführenden Transformation und konnte für diesen Anwendungsfall auf 2^{23} gesetzt werden.

$$M' = M \cdot a \quad \text{mit} \quad a = 2^{23}$$

M' kann nun als Integerarray abgespeichert werden. Dieses repräsentiert Festkommazahlen mit 9 Ganzzahl-Bits und 23 Nachkomma-Bits. Die Berechnung der Koordinaten ändert sich damit jedoch nicht, da die Division der beiden Summen von Festkommazahlen eine normale Ganzzahl (keine Festkommazahl) ist. Die folgende Rechnung zeigt die Umformung für die X-Koordinate:

$$\begin{aligned} x_{src} &= \frac{M'_{11}x + M'_{12}y + M'_{13}}{M'_{31}x + M'_{32}y + M'_{33}} \\ &= \frac{aM_{11}x + aM_{12}y + aM_{13}}{aM_{31}x + aM_{32}y + aM_{33}} \\ &= \frac{a \cdot (M_{11}x + M_{12}y + M_{13})}{a \cdot (M_{31}x + M_{32}y + M_{33})} \\ &= \frac{M_{11}x + M_{12}y + M_{13}}{M_{31}x + M_{32}y + M_{33}} \end{aligned}$$



Markierte Pixel besitzen anderen Ursprung durch Festkommarechnung mit M' als bei Fließkommarechnung mit M. Die unteren Ecken liegen außerhalb des Quellbildes.

Abbildung 4.3: Fehlerkarte der Festkomma-Transformation

Die Berechnung mit Festkommazahlen führt zu Rundungsfehlern. Abbildung 4.3 zeigt für jeden Pixel des Zielbildes, ob die Festkommarechnung einen anderen Ursprungspixel als die Fließkommarechnung liefert. Die Differenz der Ursprungspixel ist in allen Fällen nur eine Verschiebung um einen Pixel. Die Y-Koordinate weicht in zwei Bildzeilen ab. Diese liegen jedoch im unteren Bereich des Bildes, wo die Informationsdichte im Quellbild geringer ist. Dadurch ist der Fehler zu vernachlässigen.
Allgemein konzentrieren sich die Fehler auf die untere, rechte Bildecke. Dies lässt sich auf die akkumulierten Fehler der Koeffizienten von M' zurückführen. Bezogen auf das gesamte Zielbild haben 2.7% der Pixel einen anderen Ursprungspixel verglichen mit der Fließkommarechnung.

4.1.1 Kalibrierung

Die in der Transformation verwendete Matrix M wird mittels OpenCV auf dem Entwicklungsrechner ermittelt. Dafür werden jeweils vier Quell- und Zielbildpunkte bestimmt. Aus den acht Punkten wird mit der OpenCV Funktion `getPerspectiveTransform()` die inverse Matrix M^{-1} der Transformation erstellt (Codebeispiel 4.3).

Zur Kalibrierung wird ein Schachbrettmuster mit 3cm x 3cm großen Quadraten verwendet. Über die Telemetrieanwendung wird das untransformierte Bild der Kamera übertragen. Das Fahrzeug wird mittig zum Schachbrettmuster so positioniert, dass in der untersten Bildzeile die ersten Quadrate des Schachbrettmusters anfangen. Anschließend werden die Eckkoordinaten von einem Feld über 5x6 Quadrate des Schachbrettmusters aus dem Kamerabild abgelesen. Dies sind die vier Quellbildpunkte. Die vier Zielbildpunkte sind abhängig von der Auflösung des Zielbildes und dem gewünschten Verhältnis zwischen Pixel und Millimeter. Der Abstand von Hinterachse zum ersten Quadrat des Schachbrettmusters wird für den Pure Pursuit Algorithmus (Kapitel 4.4.1) benötigt und muss daher gemessen werden.

```
1  #include <opencv2/imgproc.hpp>
2  #include <iostream>
3  int main()
4  {
5      cv::Mat1f src = (cv::Mat1f(4,2) <<
6          400 -  81, 21,  // oben links
7          400 - 288, 99,  // oben rechts
8          400 + 288, 99,  // unten rechts
9          400 +  81, 21); // unten links
10     cv::Mat1f dst = (cv::Mat1f(4,2) <<
11          85,  60,       // oben links
12          85, 240,       // oben rechts
13         235, 240,       // unten rechts
14         235,  60);      // unten links
15     auto m = cv::getPerspectiveTransform(src, dst);
16     std::cout << m.inv() << std::endl;
17     return 0;
18 }
```

Codebeispiel 4.3: Berechnung Transformationsmatrix

4.2 Fahrbahnerkennung

Die Fahrbahnerkennung ist Grundlage für alle weiteren Features, welche erkannt werden sollen, daher muss der Algorithmus robust sein. Es kann nicht davon ausgegangen werden, dass alle drei Fahrbahnmarkierungen sichtbar sind, denn Hindernisse können sie verdecken und in Kurven können sie außerhalb des Sichtfeldes liegen. Der zu entwickelnde Algorithmus zur Linienerkennung muss dies folglich kompensieren.

4.2.1 Algorithmus der Linienerkennung

Der Algorithmus iteriert zeilenweise von unten nach oben über das Bild in Vogelperspektive. In jeder Zeile werden alle drei Markierungen gesucht. Für die erste zu untersuchende Zeile wird die Fahrbahnposition des letzten Frames verwendet. Im Falle eines Resets wird vermutet, dass das Fahrzeug auf der rechten Fahrspur steht. Für alle weiteren Zeilen wird davon ausgegangen, dass die Markierungen in der Nähe der gefundenen Position dieser Markierung aus der vorherigen Zeile liegt. Diese Positionen werden als Vermutung an die Markierungssuche übergeben.

4.2.2 Markierungssuche

Die Suche nach einer Markierung fängt mit einer Vermutung über die Position der Markierung an. Von dieser Vermutung aus wird gleichzeitig nach rechts und links gesucht. Es wird in der Bildzeile nach dem Pattern „dunkel, hell, . . . , hell, dunkel" gesucht, wobei es eine Obergrenze für die Anzahl von hellen Pixeln gibt. Für dunkle Pixel gibt es einen oberen Schwellwert und für helle Pixel gibt es einen unteren Schwellwert der Helligkeit.

Abbildung 4.4 zeigt den Zustandsautomat, welcher für die Suche verwendet wird. Es wird je ein Automat für die Suche nach links und für die Suche nach rechts verwendet. Zuerst wird auf einen dunklen Pixel gewartet, damit nicht mitten in einer hellen Fläche angefangen wird zu suchen. Nachdem der erste dunkle Pixel gefunden wurde wird auf helle Pixel gewartet und alle weiteren dunklen Pixel werden ignoriert. Wird ein heller Pixel gefunden, wird in den Zustand *zähle hell* gewechselt. Dort werden solange helle Pixel gezählt, bis erneut ein dunkler Pixel auftaucht, oder die Obergrenze der hellen Pixel erreicht wurde. Das Maximum soll verhindern, dass größere helle Flächen als Markierung erkannt werden.

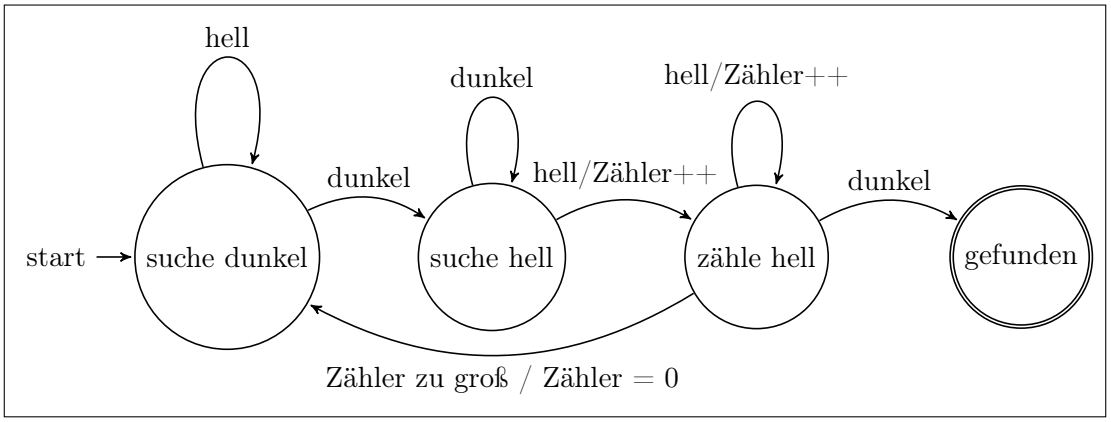


Abbildung 4.4: Zustandsautomat Markierungssuche

Damit die Markierung auch entdeckt werden kann, wenn die Vermutung genau richtig ist, fängt die Suche für beide Richtungen vor der Vermutung an.

```
1 Vermutung:                 V
2 Pixel:           LLLLLLLLLHHHLLLLLLLLLLLLL
3 SucheLinks:      <<<<987654321
4 SucheRechts:             123456789>>>>>>>>>
```

Die Suche ist beendet, wenn die erste valide Markierung gefunden wurde oder abgebrochen, wenn bei der Suche die maximale Entfernung von der Vermutung erreicht wurde. Wird in beiden Richtungen gleichzeitig eine Markierung gefunden, so wird die rechte Markierung genommen.

4.2.3 Fehlstellenkompensation

Nachdem alle drei Fahrbahnmarkierungen gesucht wurden, werden die nicht gefundenen Markierungen aus den gefundenen Markierungen berechnet. Eine fehlende mittlere Markierung kann entweder aus beiden äußeren Markierungen oder aus nur einer äußeren Markierung und der zuletzt gemessenen Fahrbahnbreite berechnet werden. Eine fehlende äußere Markierung benötigt die andere äußere Markierung. Bei zusätzlich vorhandener mittlerer Markierung wird diese für die Fahrbahnbreitenberechnung verwendet. Ansonsten wird die ermittelte Breite der vorherigen Bildzeile verwendet.

4.3 Ausweichen von statischen Hindernissen

Damit Hindernisse umfahren werden können müssen diese zuvor erkannt werden. Dafür wird in der Mitte der erkannten Fahrspur nach hellen Pixeln gesucht. Wenn die Summe der gefundenen Pixel größer als der Schwellwert für Objekte ist, dann ist ein Objekt erkannt worden. Auch Haltelinien liegen in dem Bereich, wo nach Hindernissen gesucht wird. Dies ist bei der Wahl des Schwellwertes zu bedenken, damit diese nicht fälschlicherweise als Hindernisse detektiert werden.

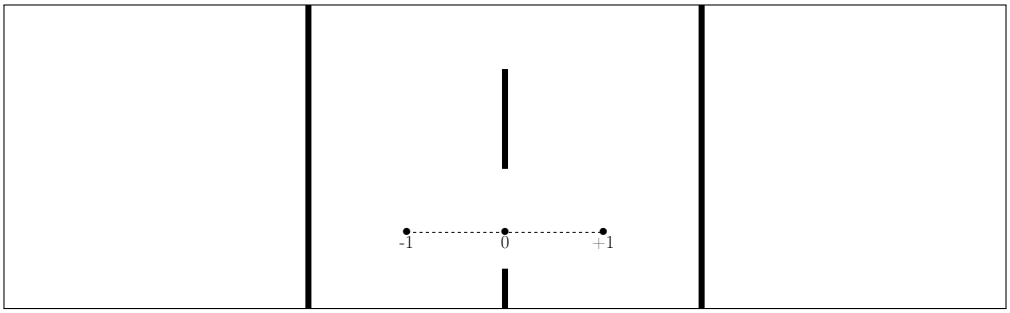
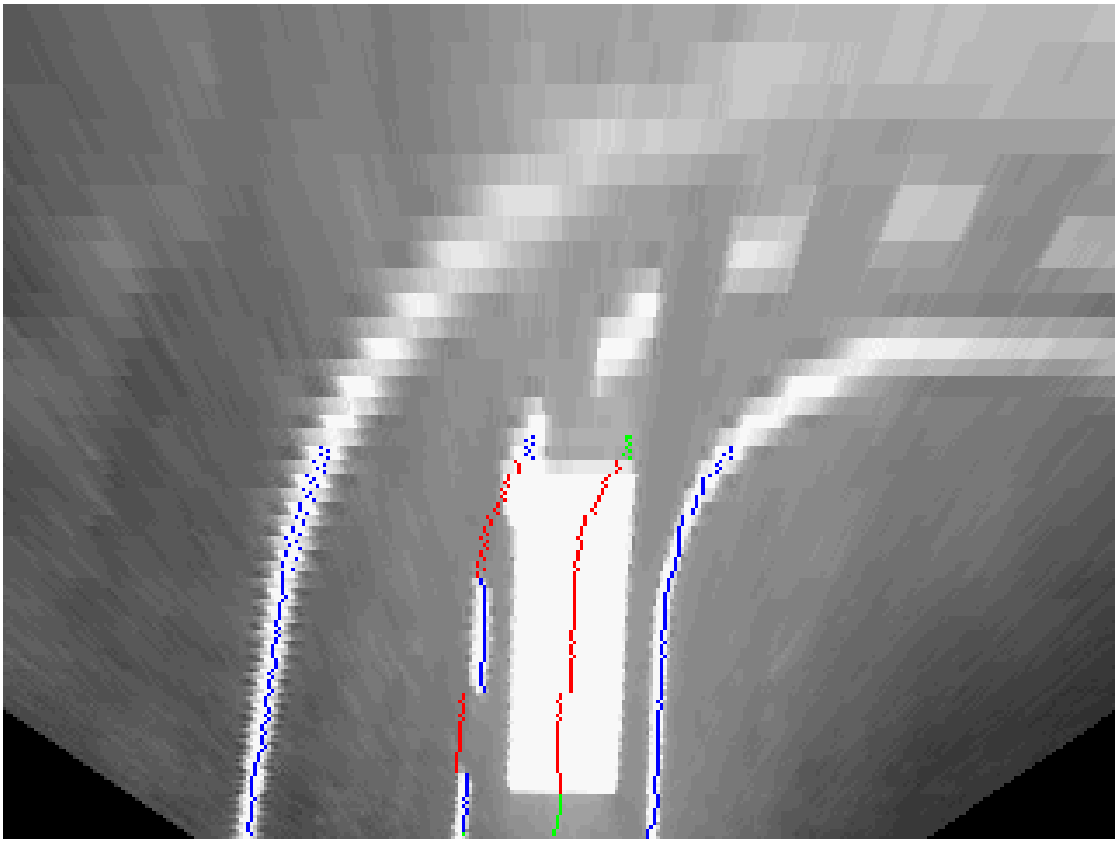


Abbildung 4.5: Fahrbahnpositionen

Das Umfahren des Hindernisses wird durch einen Spurwechsel umgesetzt. Die Variable `nav.laneSelect` gibt die anzusteuernde Position relativ zur Straße an (Abb. 4.5). Der Wert -1 bedeutet das Ziel liegt mittig auf der linken Fahrspur und der Wert 1 bedeutet das Ziel liegt mittig auf der rechten Fahrspur. Die Werte dazwischen entsprechen der linearen Interpolation zwischen den beiden Positionen. Somit bedeutet der Wert 0, dass das Ziel die Mitte der gesamten Fahrbahn ist.

Zur Vereinfachung gehen wir davon aus, dass in den Testszenarien eine blockierte Straße durch Hindernisse nicht vorkommen wird. Da das Rechtsfahrgebot eingehalten werden soll, reicht das Erkennen von Hindernissen auf der rechten Fahrspur (Abb. 4.6). Im Normalfall fährt das Fahrzeug auf der rechten Fahrspur und bei einem erkannten Hindernis auf dieser Fahrspur wird ein Spurwechsel initiiert. Das Fahrzeug bleibt so lange auf der linken Fahrspur, bis es an dem Hindernis vorbeigefahren ist. Anschließend wird wieder ein Spurwechsel zurück auf die rechte Fahrspur durchgeführt.



Fahrbahnmarkierung(blau: gefunden, rot: berechnet)
Hindernissuche (grün: frei, rot: blockiert)

Abbildung 4.6: Hinderniserkennung

Hindernisse, welche sich direkt neben dem Fahrzeug befinden, sind nicht in dessen Sichtfeld (Abb. 4.2). Daher muss der Spurwechsel zurück zur rechten Fahrspur verzögert ausgeführt werden. Um diese Verzögerung zu realisieren werden ein Tiefpass und eine Beschränkungsfunktion verwendet. Eingangssignal in die Berechnung ist, ob ein Hindernis auf der rechten Fahrspur erkannt wurde. Das Ausgangssignal ist `nav.laneSelect` im Interval $[-1; 1]$. Das Eingangssignal Signal liegt im Interval $[0; 1]$ und wird mit einem Skalierungsfaktor $-s$ und Offset 1 auf das Interval $[1 - s; 1]$ skaliert. Anschließend wird der Tiefpass auf das skalierte und verschobene Signal angewendet und zuletzt auf das Interval $[-1; 1]$ beschränkt (Abb. 4.7).

Die Variable `nav.laneSelect` wird über folgende Formel berechnet:

$$laneSelect = clamp(\quad lpf(1 - s \cdot obstacleDetected_{right}) \quad , -1, 1)$$

Über die Wahl des Skalierungsfaktors und des Tiefpasses kann die Dauer des Spurwechsels sowie die Länge der Verzögerung für den zweiten Spurwechsel zurück auf die rechte Fahrspur bestimmt werden.

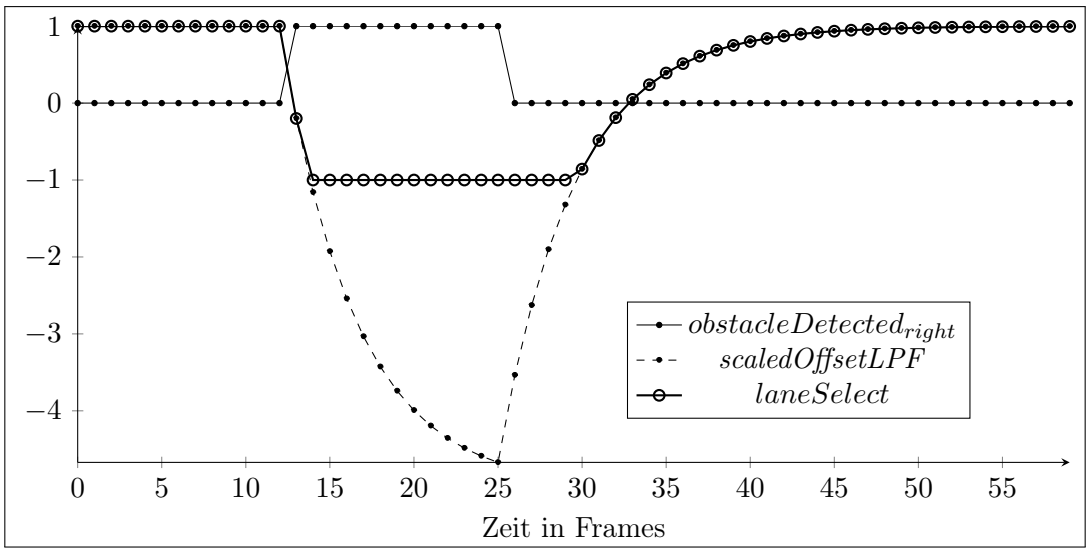


Abbildung 4.7: Funktion der Spurwahl

4.4 Steuerung des Fahrzeugs

Die Steuerung hat als Eingangssignal die relativen Koordinaten zur Fahrzeughinterachse eines Zielpunktes. Dieser wird berechnet über die erkannte Fahrbahn und die Fahrspurwahl `laneSelect`. Das Ausgangssignal für die Lenkung hat den Wertebereich -100% bis 100%. Zwei unterschiedliche Ansätze für die Steuerung werden anschließend aufgezeigt.

4.4.1 Pure Pursuit

Der Pure Pursuit Ansatz [7] berechnet für eine Zielkoordinate den Lenkradius, um diese Zielkoordinate zu erreichen. Zur Vereinfachung wird ein Einspurmodell des Fahrzeugs angenommen (Abb. 4.8). Der Ursprung für die Koordinaten liegt in der Mitte der Hinterachse. Die Y-Achse zeigt in Fahrtrichtung und die X-Achse zur Rechten des Fahrzeugs.

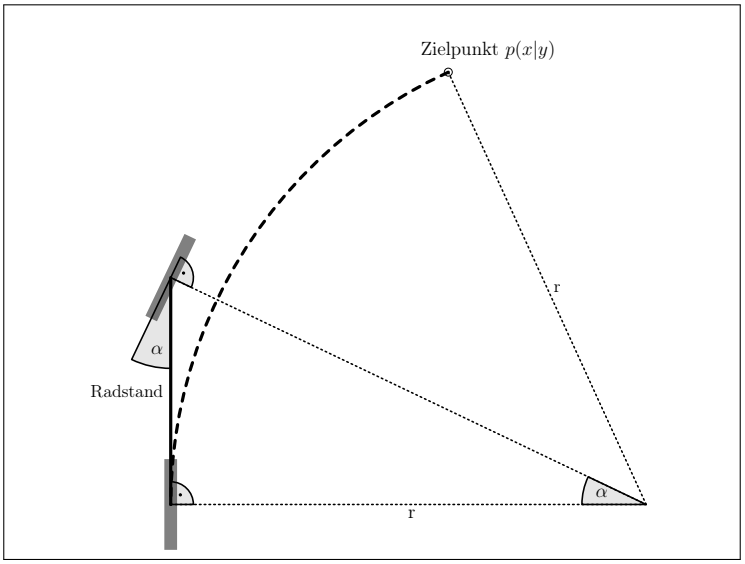


Abbildung 4.8: Einspurmodell

Der Radius r der Kreisbahn, auf der sich der Zielpunkt p befindet, wird mit folgender Formel berechnet.

$$r = \frac{p_x^2 + p_y^2}{2 \cdot p_x}$$

Der Lenkwinkel α entspricht dem Winkel zwischen der Verlängerung der Hinterachse und dem Strahl von der Lenkachse zum Kreisbahn Mittelpunkt.

$$\alpha = tan^{-1}\left(\frac{Radstand}{r}\right)$$

Für die Berechnung des resultierenden Lenkwertes in Prozent wird der Sinus des Lenkwinkels im Verhältnis zum Sinus des maximal möglichen Lenkwinkels des Fahrzeugs verwendet (Abb. 4.9).

$$steer_{percent} = \frac{sin(\alpha)}{sin(\alpha_{max})}$$

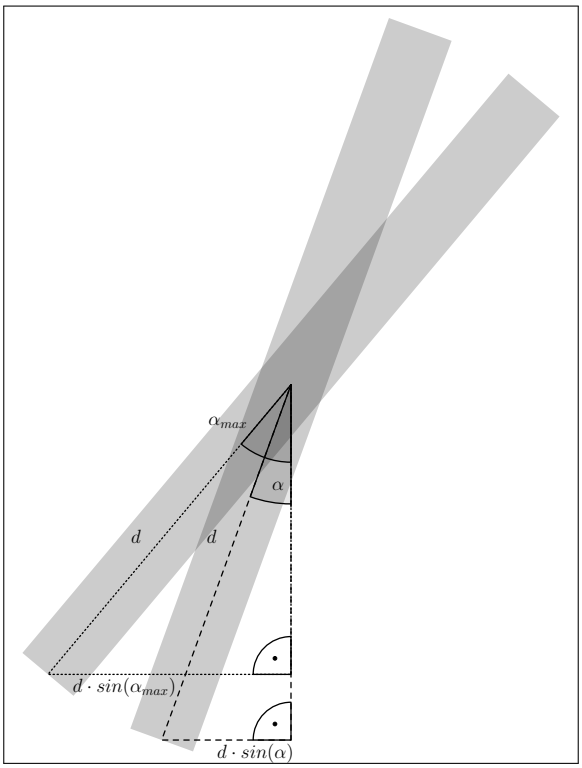


Abbildung 4.9: Lenkung: Servoweg

Die Wahl der Entfernung der Zielkoordinate beeinflusst die Reaktion der Steuerung. Liegt das Ziel weit entfernt vom Fahrzeug, so verändern sich die Stellwerte der Steuerung in geringerem Maße, was zu einem ruhigeren Fahrverhalten führt. Die Fahrspur kann jedoch verlassen werden, wenn das Ziel in einer Kurve liegt während das Fahrzeug noch vor der Kurve steht. Das Fahrzeug schneidet dann die Kurve. Mit näheren Zielkoordinaten

bleibt das Fahrzeug besser auf der Fahrspur, jedoch kann es bei höheren Geschwindigkeiten zu Oszillationen kommen, da kleine Veränderungen der Zielkoordinaten zu größeren Steuerwerten führen als bei weiter entfernten Koordinaten.

4.4.2 Lineare Verschiebung

Dieser Ansatz zur Berechnung des Stellwertes für die Lenkung verwendet den Versatz der gefundenen Fahrbahn in der untersten Bildzeile. Der Versatz wird mit einem Skalierungsfaktor skaliert. Als Skalierungsfaktor wird die Steigung der Pure-Pursuit-Funktion für Zielpunkte in der untersten Bildzeile verwendet (Abb. 4.10). Die Steigung entspricht dem Skalierungsfaktor 2.1. Das bedeutet, bei einem Versatz von mehr als 48mm erreicht die Lenkung den Vollausschlag. Für Versätze geringer als 20mm verhält sich dieser Ansatz sehr ähnlich zum Pure Pursuit Algorithmus.

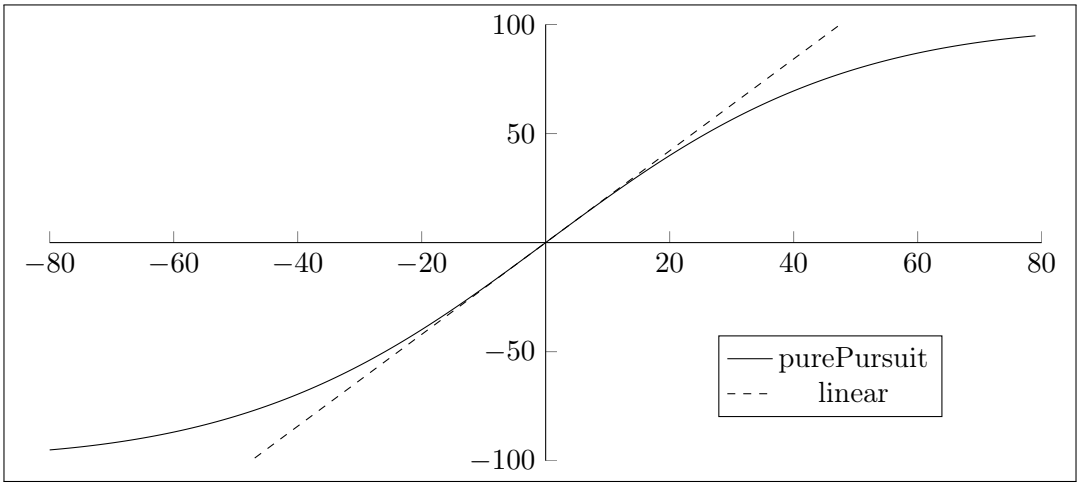


Abbildung 4.10: Linear vs Pure Pursuit

5 Evaluation

In diesem Kapitel wird sowohl die Testumgebung für Livetests mit dem uAC beschrieben als auch die Simulation für das Testen der Algorithmen zur Fahrbahnerkennung aus dem vorigen Kapitel. Des Weiteren werden Eigenschaften des Gesamtsystems beschrieben. Abschließend wird ein Livetest zur Wettkampfsimulation durchgeführt und ausgewertet.

5.1 Testumgebung

Die Umgebung, in welcher sich das Fahrzeug orientieren können soll, ist angelehnt an das Regelwerk des Carolo-Cup 2019 [1], jedoch skaliert auf ein Verhältnis von ca. 1:63 statt 1:10. Der zu befahrene Untergrund ist schwarz und die Markierungen bestehen aus weißem Klebeband. Die Abmessungen der Markierungen befinden sich in Tabelle 5.1 und die verwendete Teststrecke ist in Abbildung 5.1 zu sehen. Optische Reflexionen von Leuchtmitteln werden bestmöglich in der Testumgebung verhindert. Die Hindernisse werden durch rechteckige weiße Flächen dargestellt.

Maß	Carolo-Cup (1:10)	Testumgebung (1:63)
Fahrbahnmarkierung	18mm - 20mm	2mm
Segmentlänge Mittelmarkierung	200mm	33mm
Fahrbahnbreite	350mm - 450mm	66mm
Minimaler Kurveninnenradius	1000mm	160mm
Hindernisbreite	100mm - 400mm	16mm
Hindernislänge	min. 100mm	76mm

Tabelle 5.1: Testumgebungsmaße

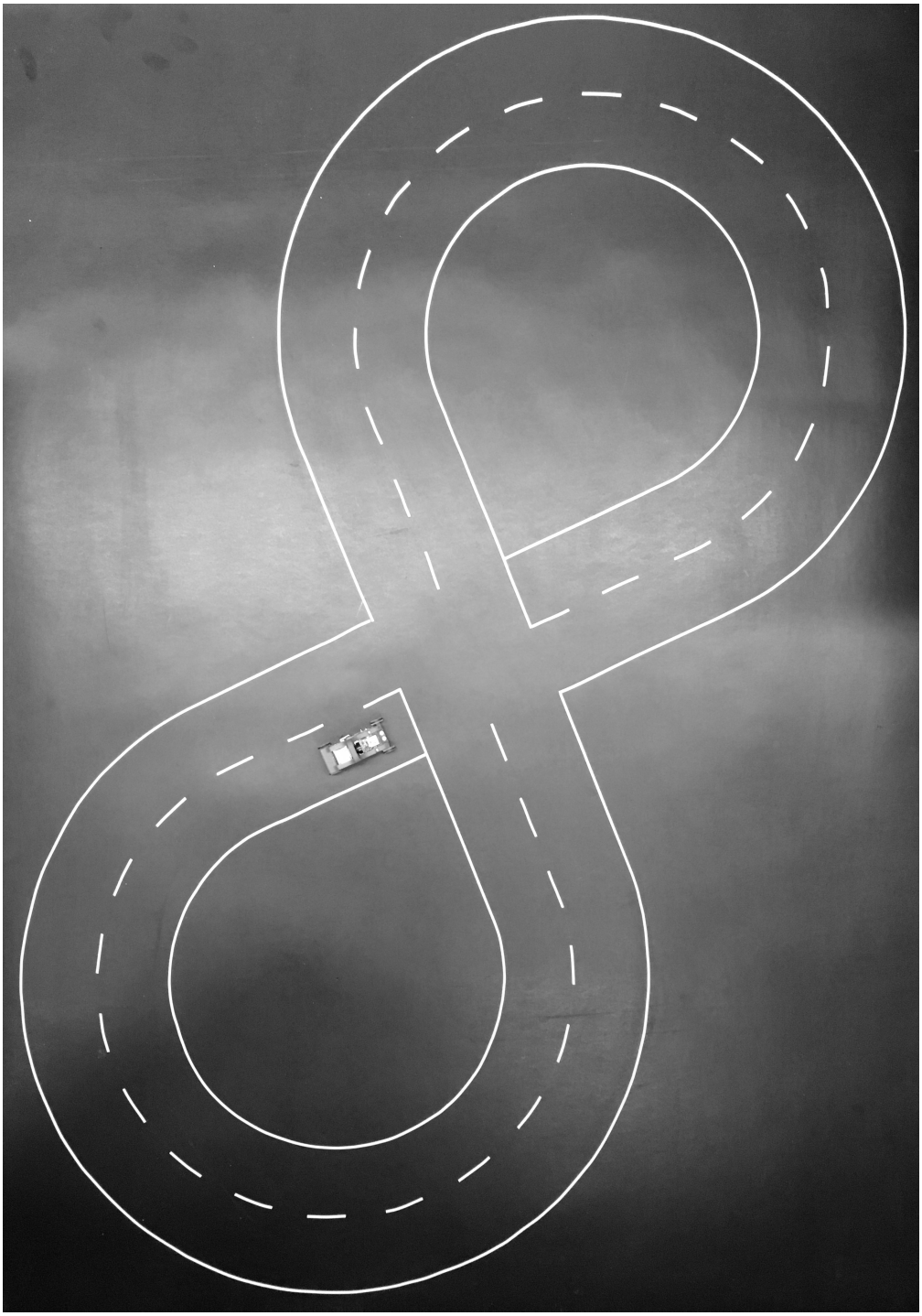


Abbildung 5.1: Teststrecke aus der Vogelperspektive

5.2 Simulation

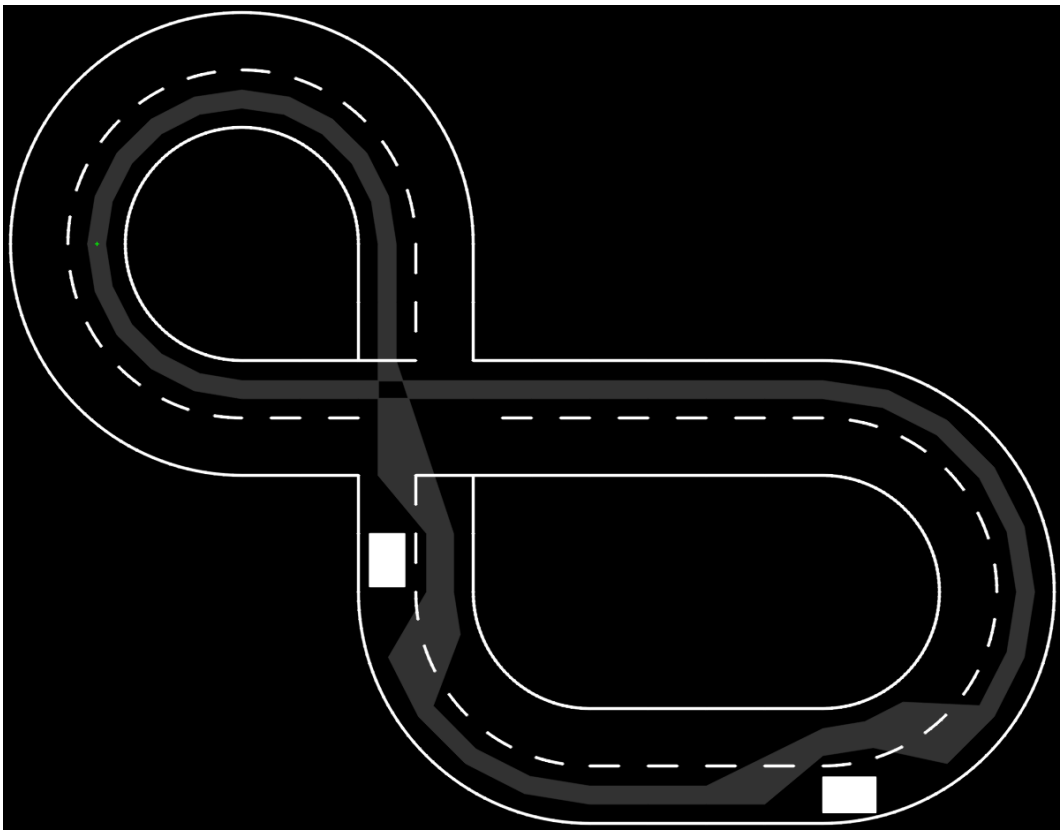
Der Vergleich und die Bewertung von verschiedenen Algorithmen zur Erkennung der Fahrbahn oder deren Parametrisierung ist unter realen Bedingungen schwierig. Situationen können nicht exakt reproduziert werden, da sie z.B. durch äußere Einflüsse verändert werden. Um das Verhalten von Algorithmen besser testen zu können, wurde eine Simulation erstellt. Diese Simulation versucht das Verhalten des Fahrzeugs in einem vereinfachten Modell nachzustellen. Verschiedene Teststrecken können simuliert werden. Das simulierte Fahrzeug bewegt sich auf der Teststrecke, während das korrespondierende, in die Vogelperspektive transformierte Kamerabild der simulierten Teststrecke erzeugt wird. Die Simulation ist in 2D und verwendet 3x3 Matrizen, um zwischen verschiedenen Koordinatensystemen zu transformieren. Zum Rendern der Bilder und für die Berechnung der Matrizen wird OpenCV verwendet.

5.2.1 Teststrecken

Die Teststrecken der Simulation werden aus Segmenten zusammengesetzt (Abb. 5.2), welche in einem Streckenobjekt gespeichert werden. Die Sequenz der Segmente kann über eine Zeichenkette angegeben werden. Kurven werden mit den Buchstaben ‚r/R' für Rechtskurven oder ‚l/L' für Linkskurven beschrieben. Kleine Buchstaben stehen für Kurven mit einem kleinen Radius und große Buchstaben für Kurven mit großem Radius. Der Buchstabe ‚s' beschreibt ein gerades Stück Straße. Mit dem Buchstaben ‚x' kann eine Kreuzung dargestellt werden. Dies entspricht den Haltelinien für beide Richtungen auf der rechten Fahrspur. Damit die Kreuzung vollständig ist, muss die vorherige oder nachfolgende Sequenz senkrecht auf das Kreuzungsstück treffen. Mit einem Leerzeichen kann dann die Kreuzung übersprungen werden, damit der Kreuzungsbereich keine Linien enthält. Der Buchstabe ‚o' erstellt ein Hindernis auf der rechten Fahrspur im nachfolgenden Segment. Wenn die Strecke kein Rundkurs ist, so kann ein Zielbereich mit dem Buchstaben ‚f' erstellt werden. Kommata oder andere zuvor nicht erwähnte Zeichen können zur Trennung einzelner Abschnitte verwendet werden, da sie ignoriert werden.

Während des Erstellens der Strecke wird zugleich der Idealbereich für das Fahrzeug berechnet (grau hinterlegt in Abb. 5.2). Dies ist der Bereich auf der Strecke, in dem sich die Hinterachse des Modellfahrzeugs bewegen soll. Das Verlassen dieses Bereichs wirkt sich negativ auf die Bewertung der simulierten Fahrt aus. Das Streckenobjekt enthält weiterhin die minimalen und maximalen Koordinaten der Streckensequenz. Diese

Koordinaten werden verwendet, um eine Übersichtskarte der Teststrecke zu erstellen, in der die Einzelschritte der Simulation eingezeichnet werden.



Sequenz: `rrrrr,rrrrrssxsoslllllssssollllll,lllllssssss ssrrrrr`

Abbildung 5.2: Simulierte Teststrecke

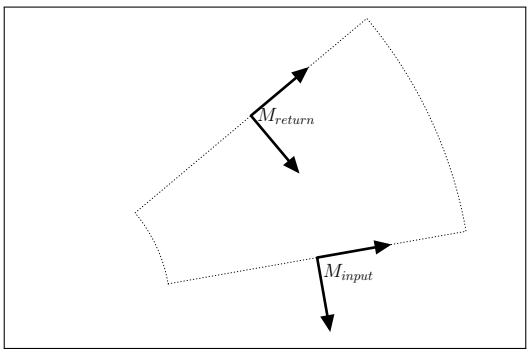


Abbildung 5.3: Koordinatensysteme beim Segmentzeichnen

Für jeden Segmenttyp gibt es eine Funktion, welche diesen Typ in ein beliebiges Bild einzeichnet. Zusätzlich zum Zielbild wird auch eine Transformationsmatrix M_{input} angegeben. Die Matrix kann eine Kombination von Verschiebungen, Drehungen und Skalierungen sein. Die Funktion gibt eine Matrix M_{return} zurück, welche den Endpunkt des Segmentes und dessen Orientierung angibt (Abb. 5.3). M_{return} kann als M_{input} für das nachfolgende Segment verwendet werden, wodurch das Segment nahtlos anschließt (Codebeispiel 5.1).

```
1  //no offset, rotation or scaling
2  Mat startTransform = Mat::eye(3,3, CV_64F);
3  // (  1 0 0
4  //    0 1 0
5  //    0 0 1  )
6  Img image;
7  Mat currentTransform = drawStraight(image, startTransform);
8  //currentTransform contains offset by one straight segment
9  currentTransform = drawRight(image, currentTransform);
10 //currentTransform contains offset+rotation of one straight segment followed
       by one right curve segment
```

Codebeispiel 5.1: Segmente Zeichnen

Das globale Koordinatensystem hat die Standardbasis. Alle Koordinaten sind so angegeben, dass eine Einheit einem Millimeter entspricht. So sind die Punkte $p_1(0,0)$ und $p_2(0,4)$ im globalen Koordinatensystem 4mm entfernt. Soll die Teststrecke beispielsweise mit einer Skalierung von 0.5mm pro Pixel gezeichnet werden, so übergibt man der Zeichenfunktion die Matrix $M_{0.5mm}$, in der die X und Y Koordinaten auf 50% skaliert sind.

$$M_{0.5mm} = \begin{pmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

5.2.2 Fahrzeug

Das simulierte Fahrzeug bildet die 3D-gedruckte Plattform ab. In diesem Modul müssen Position, Ausrichtung und Bewegungsverhalten des Fahrzeugs berechnet werden.

Position und Ausrichtung

Die Position des Fahrzeugs ist über die Mitte der Hinterachse definiert. Zusätzlich hat das Fahrzeug auch eine Ausrichtung. Diese beiden Informationen werden in einer 3x3 Matrix M_{state} gespeichert. M_{state} beschreibt das Fahrzeugkoordinatensystem. Der Sichtbereich des Fahrzeugs wird über den Abstand zur Hinterachse, seine Höhe und Breite in Pixeln, sowie dem Skalierungsfaktor Millimeter pro Pixel angegeben. Die Matrix $M_{viewOffset}$ enthält die Transformation von Pixelkoordinaten im Sichtbereich zum Fahrzeugkoordinatensystem. Die Kombination beider Transformationen M_{view} liefert das Koordinatensystem des Sichtbereichs bezogen auf das globale Koordinatensystem (Abb. 5.4).

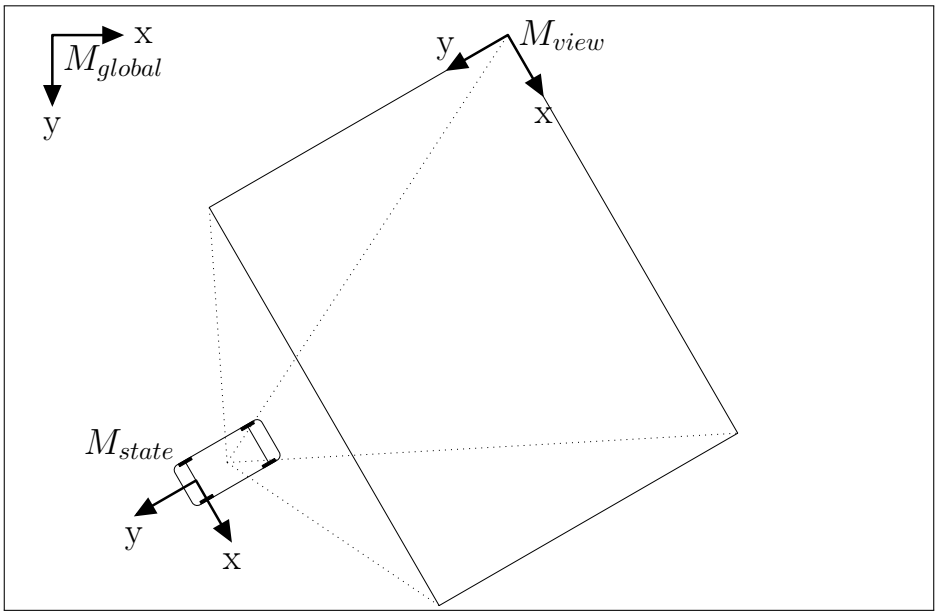


Abbildung 5.4: Koordinatensysteme der Simulation

$$M_{view} = M_{state} \cdot M_{viewOffset}$$

Jeder Punkt p_{view} im Sichtbereich kann durch eine Multiplikation mit der Matrix M_{view} in einen globalen Punkt p_{global} transformiert werden.

$$p_{global} = M_{view} \cdot p_{view}$$

Mit der inversen Matrix kann die Transformation in die andere Richtung vollzogen werden.

$$p_{view} = M_{view}^{-1} \cdot p_{global}$$

Zum Erstellen des simulierten Kamerabildes in Vogelperspektive wird die zuvor beschriebene Zeichenfunktion der Strecke verwendet. Als Transformationsmatrix wird die inverse Matrix M_{view}^{-1} übergeben.

Bewegung

Nach dem die Verarbeitung eines Frames durch den zu analysierenden Algorithmus abgeschlossen ist und die neuen Stellwerte für Lenkung und Geschwindigkeit berechnet wurden, werden diese zurück an das simulierte Fahrzeug gegeben. Der Wertebereich entspricht wie beim echten Fahrzeug -100% bis 100% für Geschwindigkeit und Lenkung. Zur Berechnung der auszuführenden Bewegung des Fahrzeugs werden der Radstand, der maximale Lenkwinkel, die maximale Geschwindigkeit sowie die Framerate der Kamera verwendet. Die Bewegung entspricht nach dem Einspurmodell wie in Kapitel 4.4.1 einer Kreisbahn mit dem Mittelpunkt des Kreises auf der verlängerten Hinterachse (Abb. 5.5).

Die zurückzulegende Distanz wird berechnet durch:

$$distance = \frac{speed_{percent}}{100} \cdot \frac{speed_{max}}{fps}$$

Dabei entspricht $speed_{max}$ der maximalen Geschwindigkeit des Fahrzeugs in Millimeter pro Sekunde. Aus dem prozentualen Lenkwinkel $steer_{percent}$ wird der Einschlagwinkel α der Vorderräder berechnet:

$$\alpha = sin^{-1}\left(\frac{steer_{percent}}{100} \cdot sin(\alpha_{max})\right)$$

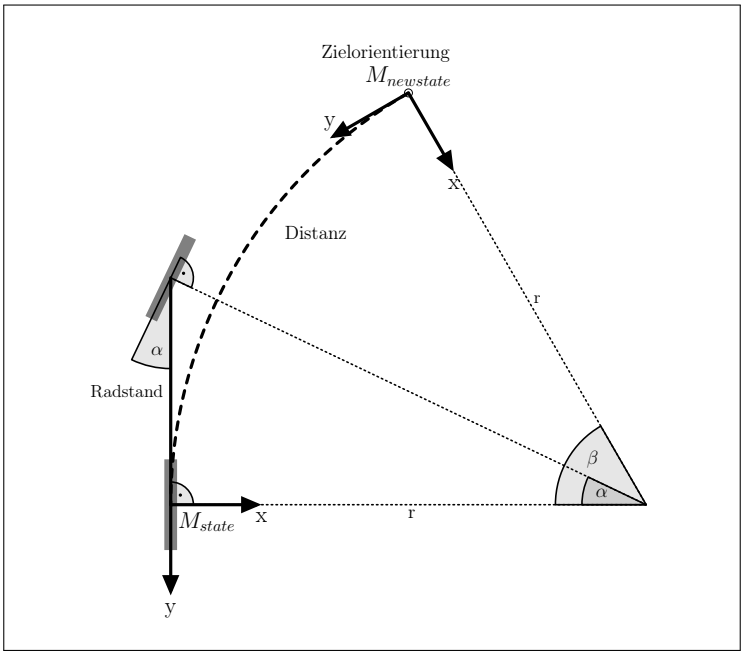


Abbildung 5.5: Bewegungsmodell

Der Radius der Kreisbahn auf der sich das Fahrzeug bewegt ist abhängig vom Radstand und dem Einschlagwinkel.

$$r = \frac{Radstand}{tan(\alpha)}$$

Der Winkel β des Kreissegmentes, der der zurückzulegenden Distanz auf der Kreisbahn entspricht, wird berechnet durch:

$$\beta = \frac{distance}{r}$$

Die resultierende Transformationsmatrix der Bewegung M_{move} hat folgende Form:

$$M_{move} = \begin{pmatrix} cos(\beta) & sin(\beta) & (1 - cos(\beta)) \cdot r \\ -sin(\beta) & cos(\beta) & sin(\beta) \cdot r \\ 0 & 0 & 1 \end{pmatrix}$$

Sie entspricht einer Rotation um β und den Punkt $(r, 0)$.

Im Sonderfall für den Steuerbefehl $steer_{percent} = 0$ entspricht die Bewegung nur der Verschiebung nach vorne.

$$M_{move} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -distance \\ 0 & 0 & 1 \end{pmatrix}$$

Abschließend wird die Position und Ausrichtung des Fahrzeugs (M_{state}) sowie die Transformation des Sichtbereichs (M_{view}) aktualisiert.

$$M_{newstate} = M_{state} \cdot M_{move}$$

$$M_{newview} = M_{newstate} \cdot M_{viewOffset}$$

5.2.3 Simulationsloop

Die Simulation funktioniert framebasiert. Für jeden simulierten Kameraframe wird ein Zyklus durchlaufen. Dafür wird zuerst das Kamerabild in Vogelperspektive erzeugt. Danach wird die Fahrbahnerkennung und Berechnung der Stellwerte ausgeführt. Als nächstes werden die Stellwerte dem Modellfahrzeug übergeben und ein Bewegungsschritt, wie im vorigen Kapitel beschrieben, wird ausgeführt. Abschließend wird die Position des Fahrzeugs mit dem berechneten Idealbereich verglichen. Befindet sich das Fahrzeug außerhalb des definierten Bereichs, so wird die Entfernung zu dem Bereich aufsummiert. Wenn sich das Fahrzeug weiter als eine Fahrbahnbreite vom Idealbereich entfernt hat oder das Fahrzeug den Zielbereich erreicht hat, wird die Simulation beendet. Die Simulation wird nach maximal zwei Minuten in simulierter Zeit gestoppt.

5.2.4 Testmöglichkeiten

In der Simulation lässt sich die Robustheit der Linienerkennung testen. Eine Möglichkeit ist, dass die Steuerwerte absichtlich reduziert werden, wodurch das Fahrzeug die Fahrbahn verlässt. Es treten somit extremere Situationen auf, wodurch weniger der Fahrbahn im Sichtbereich des Fahrzeugs liegt. In Abbildung 5.7 ist dargestellt, wie sich das Fahrzeug über die Teststrecke bewegt hat. Das rote Rechteck zeigt die Fahrzeugposition und -orientierung und das blaue Rechteck zeigt den Sichtbereich des Fahrzeugs an. Die Abbildung 5.6 zeigt den Sichtbereich des Fahrzeugs inklusive der erkannten Linien in blau, der berechneten Linien in rot und der berechneten Fahrspurmitte in grün. Von der Fahrbahn ist nur noch ein Teil der rechten Fahrspur zu sehen.

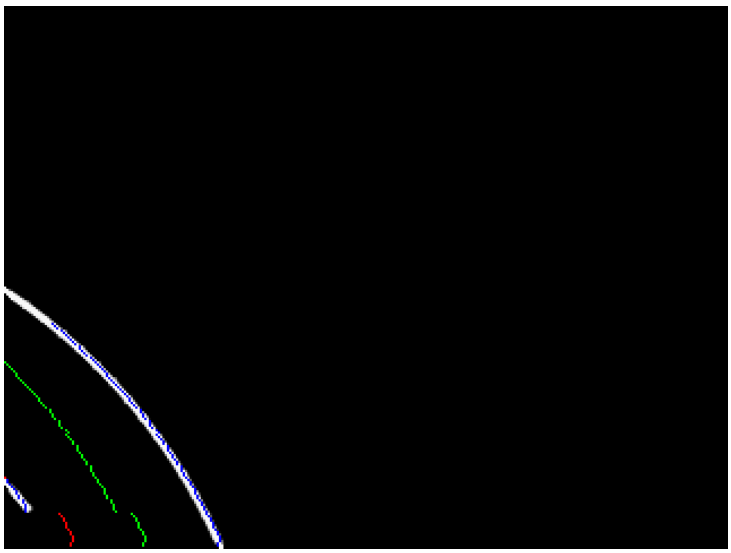


Abbildung 5.6: Simulation mit 25% der Lenkstellwerte - Fahrzeugperspektive

Als Index zur Bewertung einer simulierten Fahrt werden die akkumulierten Abstände zum Idealbereich durch die zurückgelegte Distanz geteilt und von eins abgezogen.

$$performance = 1 - \frac{\sum err}{distance_{total}}$$

Damit entspricht die Performance von 1 einer Fahrt ohne Verlassen des Idealbereichs. Außerdem wird eine Übersichtskarte mit der gefahrenen Spur der Fahrzeugposition abgespeichert.

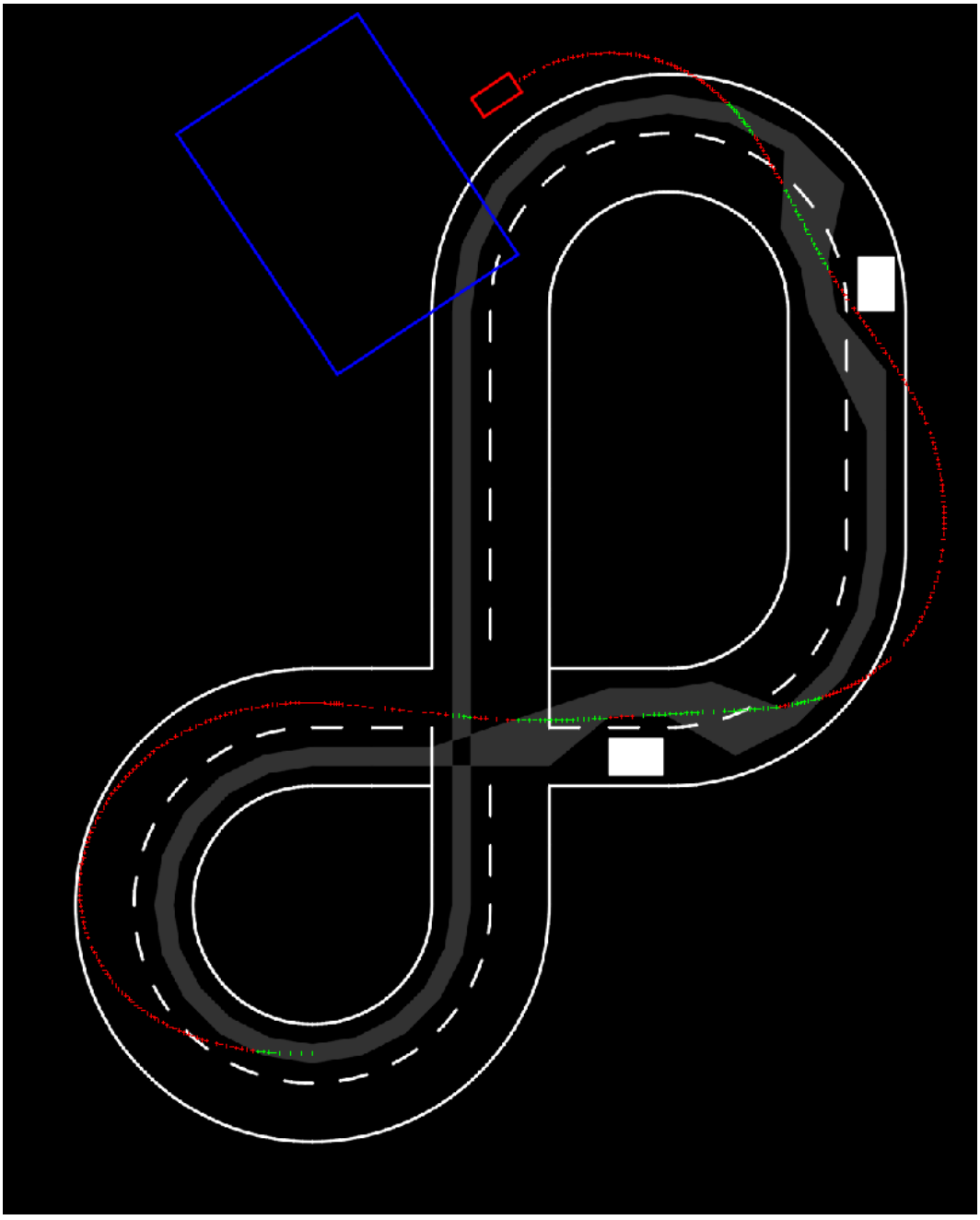


Abbildung 5.7: Simulation mit 25% der Lenkstellwerte - Übersicht

5.2.5 Einschränkungen

Das Fahrzeugmodell ist idealisiert und alle Stellwerte werden direkt erreicht. Dies entspricht nicht dem Verhalten des realen Fahrzeugs, welches eine gewisse Zeit zur Beschleunigung und zum Einstellen des Lenkwinkels braucht. Dennoch verhält sich das Modellfahrzeug sehr ähnlich zu dem realen Fahrzeug (Abb. 5.8).

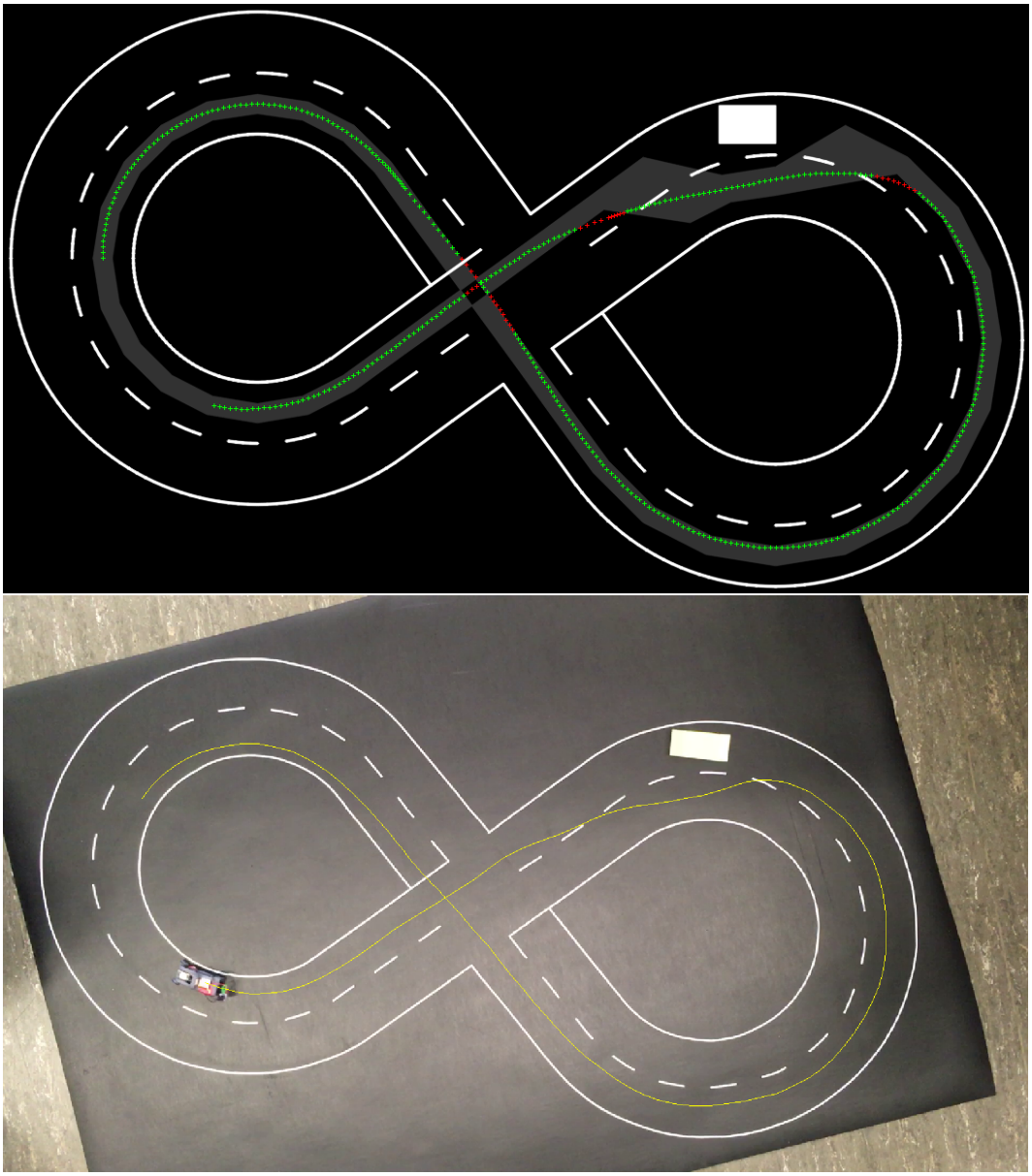


Abbildung 5.8: Vergleich Modellfahrzeug und reales Fahrzeug

Die simulierten Kamerabilder haben kein Rauschen und die Detailtiefe in der Vogelperspektive ist konstant über den gesamten Bildbereich (Abb. 5.9).

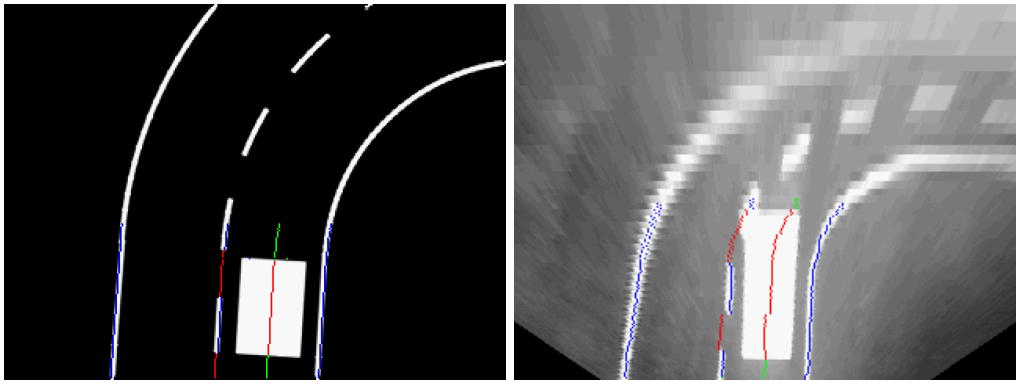


Abbildung 5.9: Vergleich Simulation und Echtbild

Der Untergrund in der Simulation ist durchgehend schwarz und alle Markierungen und Hindernisse sind durchgehend weiß. Außerdem werden keine Reflexionen oder Beleuchtungsveränderungen simuliert. Dadurch sind die Bedingungen zum Erkennen der Markierungen und Hindernisse signifikant besser als mit echten Kamerabildern.

Im Kreuzungsbereich überlagern sich die Idealbereiche der beiden Richtungen. Da der gesamte Idealbereich in einem Polygon hinterlegt ist, gehört die Schnittfläche nicht zum Idealbereich (Kreuzung in Abb. 5.2). Im Kreuzungsbereich wird daher fälschlicherweise ein Abstand zum Idealbereich berechnet, obwohl das Fahrzeug innerhalb des Idealbereichs fährt.

5.3 Eigenschaften des Micro Autonomous Car

Energie

Das uAC hat mit einer Versorgungsspannung von 3.9V einen Stromverbrauch von 192mA bei aktiver Telemetrieübertragung und Navigation, jedoch inaktiver Aktorik. Dieser Wert wurde mit einem Keysight E36313A Labornetzgerät über 50 Messpunkte ermittelt. Eine exakte Messung des Energieverbrauchs während der Fahrt, also inklusive Aktorik, war nicht ohne erhebliche Beeinflussung des Systems durch zusätzliche Fahrwiderstände möglich. Daher wurde ein Laufzeittest durchgeführt. Mit dem verbauten 100mAh Akku fährt das uAC für neun Minuten bei ca. 40% Motoransteuerung auf der Teststrecke. Sinkt

die Akkuspannung unter die Grenze, bei der der ESP32 stabil läuft, wird ein Reset im Mikrocontroller ausgelöst. Der Laufzeittest wurde mit dem ersten Reset beendet.

Rechenzeit

Für die Verarbeitung eines Kamerabildes benötigt das System inklusive Bildtransformation, Fahrbahn- und Hinderniserkennung sowie Steuerwerteberechnung 25.4ms. Weitere 42.7ms stehen zur Verfügung für weitere Algorithmen bis das Einlesen des nächsten Kamerabildes beginnt. Das Verfahren zur Messung der Zeiten entspricht dem Verfahren über GPIOs wie bei der Messung der Transformationsalgorithmen. Der Algorithmus zur Fahrbahnerkennung wurde auf die maximale Bildzeilenanzahl (240) konfiguriert für eine Situation mit maximalem Aufwand.

Telemetrie-Datendurchsatz

Die Bildübertragung der Telemetrie kann ca. sieben Bilder pro Sekunde bei 381 kByte pro Sekunde übertragen. Also wird ca. jedes zweite Bild aus der Kamera übertragen. Die Datenübertragung der Telemetrie überträgt 12.5 Mal pro Sekunde (für jeden Frame ein Mal) 216 Byte für Parameter und Variablen. In Summe sind das 2.7 kByte pro Sekunde. Während der Tests war das uAC per WLAN mit einem TP-Link AC1750 Router verbunden. Der Telemetrie Rechner nutzte eine Ethernetverbindung zum Router.

Grenzen

Während der Tests zeigten sich Grenzen des entwickelten Systems. Die korrekte Erfassung der Fahrbahn in engen Kurven kann mit dem verwendeten Algorithmus nicht erreicht werden. Durch das zeilenweise ausgeführte Scannen im Bild können keine korrekten Annahmen über die Fahrbahnbreite gemacht werden, weshalb beispielsweise die Mittelmarkierung fälschlicherweise als linke Markierung erkannt wird (Abb. 5.10). Für eine stabilere Erkennung müsste das Bild senkrecht zur bisher oder zuvor gefundenen Fahrbahn gescannt werden. So könnte dann auch von einer konstanten Fahrbahnbreite ausgegangen werden.

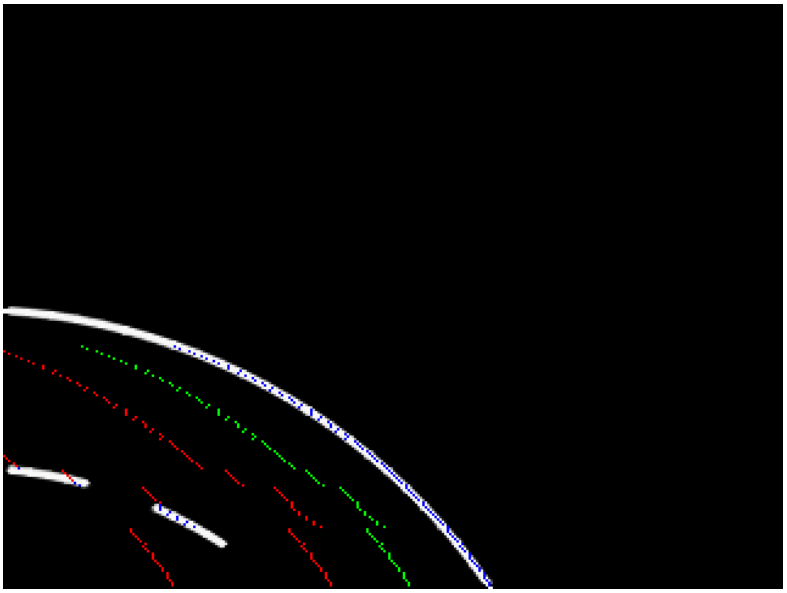


Abbildung 5.10: Simulation: Fehlerhafte Fahrbahnerkennung in engen Kurven

Reflexionen durch den glatten Untergrund und die niedrige Position der Kamera können zu Problemen bei der Fahrbahnerkennung führen, weil der Unterschied zu den Fahrbahnmarkierungen geringer wird (Abb. 5.11).

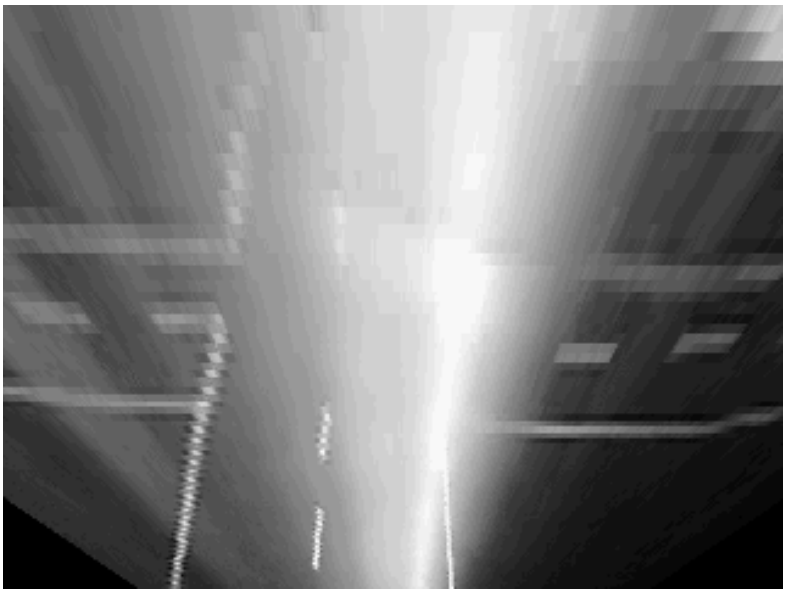


Abbildung 5.11: Reflexionen

5.4 Wettkampf

Zur Bewertung des Verhaltens der Plattform und der entwickelten Algorithmen wird eine Wettkampfsimulation als Livetest durchgeführt. Der Wettkampf stellt die Situation der Disziplin *Obstacle Evasion Course* des Basic-Cup bezogen auf den Teil *Static Obstacles* [1, S. 20] nach. Ziel ist es, mit dem Fahrzeug eine maximale Distanz innerhalb von zwei Minuten zurückzulegen. Dabei muss statischen Hindernissen ausgewichen werden. Kreuzungen dürfen anders als im Basic-Cup überfahren werden, da keine dynamischen Hindernisse auf der Strecke sind. Hindernisse werden durch 33mm x 76mm große weiße Rechtecke repräsentiert. Je zwei Hindernisse stehen in der Links- und Rechtskurve (Abb. 5.12). Am Ende der Linkskurve ist eine Lücke in der Fahrbahnmarkierung durch überkleben mit schwarzem Tape.

Es gibt Strafdistanzen von 800mm ($\sim$ 5000mm/6.3) für das Berühren (hier Überfahren) von Hindernissen und das Überfahren von Fahrbahnmarkierungen mit mehr als einem Rad.

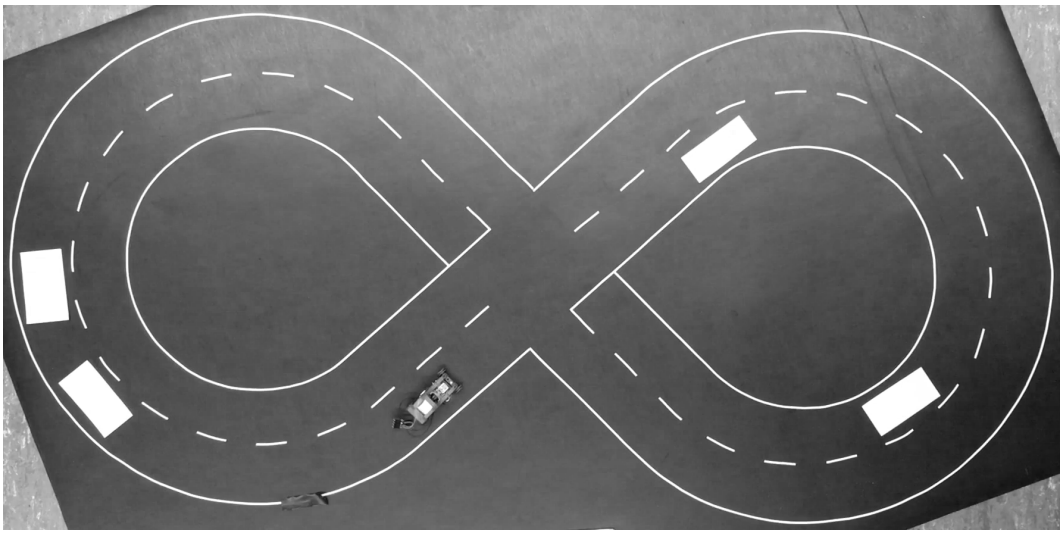


Abbildung 5.12: Wettkampfstrecke

Der mittlere Radius der Kurven ist 200mm. Die geraden Strecken inklusive der Kreuzung sind jeweils 450mm lang. Der gesamte Rundkurs hat eine Länge von:

$$1.5 \cdot (2 \cdot \pi \cdot 200mm) + 2 \cdot 450mm = 2784mm$$

5.4.1 Auswertung

In den zwei Minuten Wettfahrt hat das uAC sechs komplette Runden geschafft. Von der siebten Runde fehlte ein Halbkreis (628mm). Die zurückgelegte Strecke betrug 18.86m.

Zeit	Strafe	Strafdistanz
0:30	Linienübertritt	800mm
0:45	Hindernisberührung	800mm
1:03	Hindernisberührung	800mm
1:24	Linienübertritt	800mm
1:38	Hindernisberührung	800mm
-	Summe	4000mm

Tabelle 5.2: Strafen

Die gesammelten Strafen durch Kollisionen kamen durch das verfrühte Zurückkehren zur rechten Fahrspur zustande. Die Strafen für Linienübertritte fanden jeweils nach dem Überqueren der Kreuzung von der Seite mit Haltelinien auf. Nach Abzug der Strafdistanzen (Tabelle 5.2) von der zurückgelegten Strecke bleibt eine Distanz von 14.86m. Zur Einordnung wird das Ergebnis mit 6.3 Multipliziert und mit den Ergebnissen des Basic-Cup 2018 verglichen: 14.86m entprechen in Dimensionen des Basic-Cups einer Distanz von 93.62m.

Team	Distanz(m)	Strafdistanz(m)	Differenz(m)
uAC	119	25	94
TeamWorstCase	47	109	-62
TU-Smart	57	141	-84
Selfie	48	80	-32
RoBoss	55	152	-97
AAutonomous	13	110	-97
Die Eidgenossen	42	61	-19
OSCAR	49	109	-60
Phoenix Robotics	230	112	108
oTToCar	171	247	-76
e.Wolf	90	99	-9

Daten von der Carolo-Cup Webseite [2]

Tabelle 5.3: Ergebnisse Obstacle Evasion Course Basic-Cup 2018 und uAC

Im Vergleich zu den Teams des Jahres 2018 ist die ohne Strafen zurückgelegte Distanz des uAC die drittbeste des Feldes (Tabelle 5.3). Durch die Einschränkung auf statische

Hindernisse ist es nicht möglich gewesen Strafen bezüglich dynamischer Hindernisse zu bekommen. Außerdem konnten Kreuzungen direkt befahren werden.

Unter der Annahme, dass das Fahrzeug nur die Hälfte der Zeit fahren kann und die andere Hälfte an Kreuzungen steht, würde immer noch eine Distanz (ohne Strafen) von 59.5m erreicht werden. Damit wäre dies die viertbeste Distanz. Bei sieben Runden und einer Mindestzeit von drei Sekunden, die das Fahrzeug an der Haltelinie stehen muss, stünde das Fahrzeug insgesamt für weitere 39 Sekunden, um dynamischen Hindernissen die Vorfahrt zu gewähren.

Der Vergleich unter Berücksichtigung der Strafen ergäbe den zweiten Platz. Dieser Vergleich ist jedoch aufgrund der hohen Strafdistanzen der meisten Teams und der Einschränkungen der Aufgabe für das uAC weniger aussagekräftig und ungeeignet, um die Leistung der Plattform einzuordnen.

6 Fazit

Diese Arbeit beschäftigte sich mit der Entwicklung eines kamerabasierten minimalen autonomen Systems.

Die ausgewählten Hardwarekomponenten sowie das neu designte Fahrwerk haben sich während der Livetests bewährt. Das zum Einsatz kommende 3d-Druck-Verfahren ermöglichte das Erstellen mehrerer Testfahrzeuge mit annähernd gleichem Verhalten. Der kleinste zu erreichende Lenkradius war mit 100mm ausreichend, um in engen Kurven Überholmanöver auszuführen. Das Weitwinkelobjektiv erfasste auch in extremen Situationen, wie zum Beispiel während der Reduzierung der Lenkwerte auf 25%, immer noch eine der drei Fahrbahnmarkierungen. Lediglich die Akkuleistung mit 9 Minuten Laufzeit war etwas gering und müsste für den Einsatz zum Beispiel im Miniatur Wunderland verbessert werden.

Durch die Priorisierung von Ressourcenverbrauch, bereits in den grundlegenden Softwaremodulen, blieben ausreichende Ressourcen für die Algorithmik der Bildverarbeitung des autonomen Systems. Die Bild- und Datenübertragung der Telemetrie waren essentielle Hilfsmittel während der Entwicklung und der Inbetriebnahme der Kameraschnittstelle. Besonders während der Livetests haben sich diese als hilfreich erwiesen.

Die im Rahmen dieser Arbeit gegenübergestellten Möglichkeiten zur Optimierung und deren Bewertung zeigen, welches Potential die Optimierung für Systeme mit minimalen Ressourcen hat. Durch eine geringe Reduzierung der Genauigkeit der perspektivischen Projektion konnte fast eine Erhöhung der Geschwindigkeit um den Faktor 10 bei der Berechnung erreicht werden. Ohne diese Optimierung wäre die relativ hohe Bildrate von 12.5 fps nicht zu erreichen und müsste auf unter 6.5 fps gesenkt werden, damit die Transformation nicht mit dem Einlesen des nachfolgenden Frames kollidiert. Die durch diese Optimierung frei gewordene Rechenleistung konnte für die Algorithmen der Fahrbahnerkennung und Steuerung des Fahrzeugs verwendet werden.

Durch die in dieser Arbeit entwickelte Simulation von Teststrecken und deren transformierten Kamerabildern ist ein weiteres Werkzeug entstanden, welches bei der Parametrisierung und Evaluierung der Fahrbahnerkennung sehr hilfreich war. Die Simulation kann durch eine alternative Parametrisierung auch für die Entwicklung von Spurfindealgorithmen weiterer Wettbewerbsteilnehmer oder anderer Projektgruppen der HAW weiter verwendet werden.

In der Wettkampfsimulation hat das uAC sehr gut abgeschnitten. Die gesammelten Strafen kamen durch eine zu starke Vereinfachung der Fahrsituationen im Bereich der Kreuzung und beim Überholen zustande. In Summe waren die Strafen deutlich geringer als die der Teams des Basic-Cup 2018. Die Einschränkung auf einen Aspekt der Disziplin lässt jedoch die Frage offen, ob das entwickelte System mit den Systemen der Teams im Basic-Cup direkt konkurrieren kann.

Wie sich gezeigt hat, ist die finale Plattform geeignet für das Testen einfacher Bildverarbeitungsalgorithmen. Durch den kleinen Formfaktor bietet sich diese Plattform gut als Einstieg in das Thema der autonomen Systeme an, denn es benötigt keine großen Teststrecken: Ein Rundkurs in Form der Zahl Acht kann auf einer Fläche von 800mm x 1200mm untergebracht werden. Der niedrige Kostenfaktor von unter 100 Euro erleichtert ebenfalls den Einstieg.

Die im Vorfeld dieser Arbeit gesetzten Ziele wurden erreicht. Die Arbeit zeigt aber, dass noch nicht das gesamte Potential der Plattform ausgenutzt wurde und zeigt gleichermaßen, dass weitere technische Ansätze realisierbar sind.

Ausblick

Die Teilnahme am Basic-Cup mit dem in dieser Arbeit entwickelten Grundsystem und einem passenden Fahrwerk in 1:10 ist denkbar. Die Umsetzung eines Zustandsautomaten, welcher die Situationen des Überholmanövers und der Vorfahrt an Kreuzungen abbildet, wäre dafür eine nötige Erweiterung der Logik.

Die Umsetzung eines LKWs der Größe H0 mit der Elektronik der 3D-gedruckten Plattform sollte mit wenig Aufwand möglich sein. Mit einem Fahrwerk in H0 könnten komplizierte Verkehrssituationen und Straßenführungen beispielsweise im Miniatur Wunderland Hamburg getestet und die Möglichkeiten der kamerabasierten Minimalautonomie erforscht werden.

Die Umsetzung eines PKWs der Größe H0 ist mit den erhältlichen Platinen nicht umzusetzen. Durch ein Neudesign der Hauptplatine speziell für diesen Anwendungszweck ließe sich Platz sparen: Komponenten wie die Ladeschaltung und der USB-Serial Adapter können ausgelagert werden, denn sie sind für die Funktionalität während des autonomen Betriebs nicht nötig. Die Kombination der Schaltungen des Linearservos und des Mikrocontrollers auf einer Platine könnte außerdem eine kompaktere Gesamtplatine ermöglichen und erspart die Verwendung von Kabeln zwischen den Komponenten. Durch Integration der Lagerung für die Lenkung und Montagestellen für den Antriebsstrang könnte eine neu designte Platine zusätzlich als Bodenplatte des Fahrwerks dienen.

Im Bereich der Positionsbestimmung der autonomen Plattform gibt es weiteres Optimierungspotential: Einerseits ist die Montage der IMU MPU6050 auf der aktuellen Platine vorgesehen, wodurch die Orientierung des Fahrzeugs bestimmt werden kann. In Kombination mit der Bestimmung der Motordrehzahl ließe sich eine Schätzung der Fahzeugposition machen. Andererseits wäre der Einsatz von Simultaneous Localization and Mapping (SLAM) denkbar. Die Segmente der Mittellinie ließen sich leicht in der Vogelperspektive erkennen und als Features mit relativen Fahrzeugkoordinaten für den Algorithmus verwenden.

Ein weiteres interessantes und immer mehr in den Fokus von Forschungsgruppen rückendes Thema ist die Car-to-Car-Kommunikation: Die Funkverbindungen des ESP32s bieten vielfältige Möglichkeiten zur Kommunikation zwischen mehreren autonomen Fahrzeugen. Alternative Vorfahrtslösungen und andere Verkehrsproblemstellungen könnten damit praktisch erforscht werden.

Ein Ansatz, der ebenfalls durch die Funkverbindungen und bereits vorhandenen Telemetrieschnittstellen ermöglicht wird, wäre das Testen komplexerer Algorithmen. Die Plattform wird hierbei als verteiltes System verwendet werden: Der Kamerastream könnte von einem leistungsstarken Rechner empfangen und verarbeitet werden. Parallel könnten die berechneten Steuerwerte über die Fernsteuerung an die Plattform gesendet werden.

Das Potential der Plattform wurde noch nicht ausgeschöpft und ich bin gespannt, was mit ihr noch alles erreicht wird.

Literaturverzeichnis

[1] *Basic-Cup Regulations 2019.* https://wiki.ifr.ing.tu-bs.de/carolocup/en/system/files/180710_Basic-Cup%20Regulations.pdf. 2018. – [Zugriffsdatum: 16.05.2019]

[2] *Carolo-Cup 2018 - Ergebnisse.* https://wiki.ifr.ing.tu-bs.de/carolocup/system/files/Ergebnisse_2018.pdf. 2018. – [Zugriffsdatum: 07.06.2019]

[3] *Team-CDLC: Carolinchen X.* http://www.team-cdlc.de/index.php?site_id=caroX. 2018. – [Zugriffsdatum: 21.05.2019]

[4] *Die Software des KITcar-Teams!* https://kitcar-team.de/software.php. 2019. – [Zugriffsdatum: 21.05.2019]

[5] Carrara, Reto: *Consensus, Asynchronous lock-free Three-Buffer Data-Sharing for Embedded System using C/C++.* http://http://www.carrara.ch/fachbeitraege/Consensus.pdf. 2004. – [Zugriffsdatum: 17.06.2019]

[6] comma.ai: *open source driving agent.* https://github.com/commaai/openpilot. 2019. – [Zugriffsdatum: 24.06.2019]

[7] Coulter, R. C.: *Implementation of the Pure Pursuit Path Tracking Algorithm.* https://www.ri.cmu.edu/pub_files/pub3/coulter_r_craig_1992_1/coulter_r_craig_1992_1.pdf. 1992. – [Zugriffsdatum: 05.06.2019]

[8] Espressif Systems: *ESP32 Technical Reference Manual.* https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf. 2018. – [Zugriffsdatum: 17.05.2019]

[9] Espressif Systems: *ESP32 Series Datasheet.* https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf. 2019. – [Zugriffsdatum: 07.05.2019]

[10] Köseoğlum, M. ; Çelik, O. M. ; Pektaş, Ö.: Design of an autonomous mobile robot based on ROS. In: *IEEE 2017 International Artificial Intelligence and Data Processing Symposium (IDAP)* (2017)

[11] M5Stack: *Base espressif esp32-camera*. https://github.com/m5stack/m5stack-cam-psram. 2019. – [Zugriffsdatum: 07.05.2019]

[12] M5Stack: *Unit M5Camera*. https://docs.m5stack.com/#/en/unit/m5camera. 2019. – [Zugriffsdatum: 07.05.2019]

[13] Miniatur Wunderland: *Fahrzeug-Technik | Miniatur Wunderland Hamburg*. https://www.miniatur-wunderland.de/wunderland-entdecken/technik/carsystem/fahrzeug-technik/. 2019. – [Zugriffsdatum: 25.06.2019]

[14] OmniVision: *Advanced Information Preliminary Datasheet OV2640*. https://www.uctronics.com/download/cam_module/OV2640DS.pdf. 2006. – [Zugriffsdatum: 17.05.2019]

[15] Revell: *Mini RC Car - Cabrio, schwarz*. https://www.revell.de/produkte/revell-control/cars/mini-rc-car-cabrio-schwarz.html. 2019. – [Zugriffsdatum: 21.05.2019]

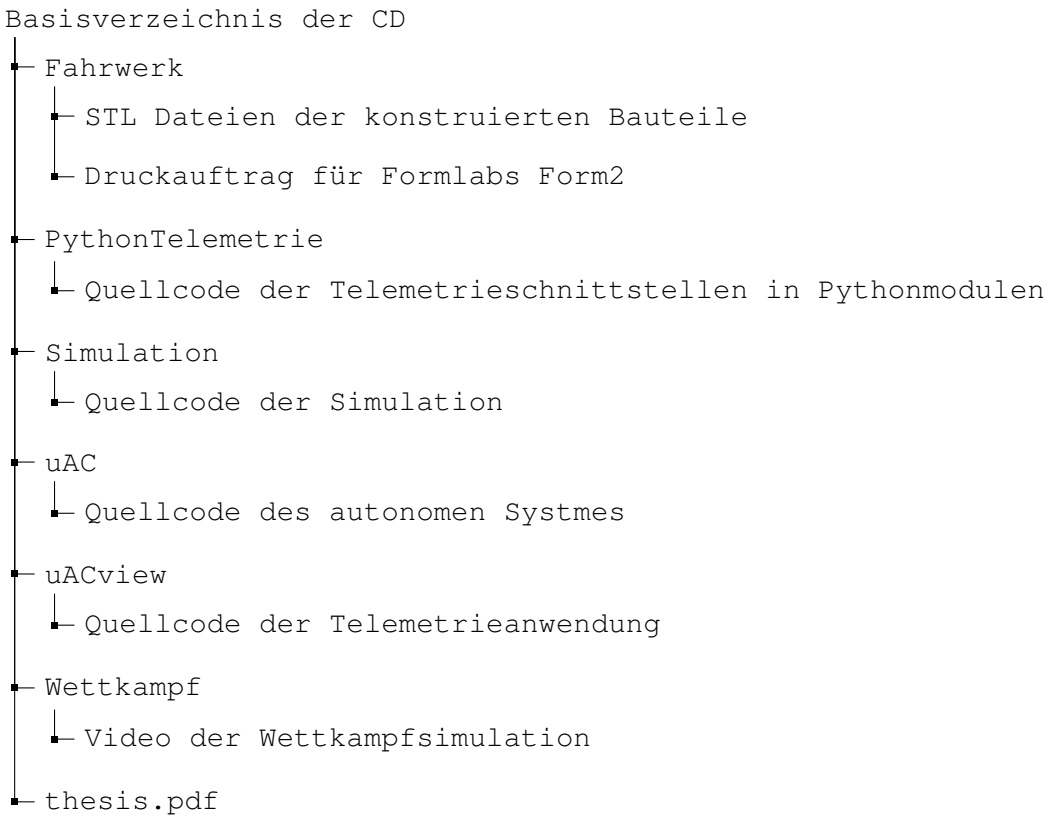
[16] Skydio: *Skydio - Technology*. https://www.skydio.com/technology/. 2019. – [Zugriffsdatum: 24.06.2019]

[17] Stark, F. ; Burau, H.: *nHAWigatora miniature ship | autosys*. https://autonomesysteme.informatik.haw-hamburg.de/platforms/2019nhawigatora/. 2019. – [Zugriffsdatum: 25.06.2019]

[18] Tesla: *Autopilot | Tesla Deutschland*. https://www.tesla.com/de_DE/autopilot. 2019. – [Zugriffsdatum: 24.06.2019]

A Anhang

A.1 Ordnerstruktur der zugehörigen CD



Glossar

Basic-Cup Wettkampf der Basisleistungsklasse innerhalb des Carolo-Cups.

Carolo-Cup Ein von der TU Braunschweig veranstalteter Wettbewerb zum autonomen Fahren.

Fahrbahn Der aus mehreren Spuren bestehende Bereich, welcher von einem Fahrzeug befahren werden darf.

Fahrspur Eine Spur der Fahrbahn.

Feature Ein erkanntes Merkmal im Bild (zum Beispiel eine erkannte Linie).

Frame Ein von der Kamera empfangenes Bild.

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Ort	Datum	Unterschrift im Original