HAW HAMBURG

MASTERTHESIS
Malte Dittmar

Kollisionsvermeidung eines mobilen Roboters mittels Schätzung der Bewegungsrichtung und Geschwindigkeit von Objekten aus 3D-Bilddaten

FAKULTÄT TECHNIK UND INFORMATIK
Department Informations- und Elektrotechnik

Faculty of Computer Science and Engineering
Department of Information and Electrical Engineering

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Malte Dittmar

Kollisionsvermeidung eines mobilen Roboters mittels Schätzung der Bewegungsrichtung und Geschwindigkeit von Objekten aus 3D-Bilddaten

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang *Master of Science Automatisierung*
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. -Ing. Marc Hensel
Zweitgutachter: Prof. Dr. -Ing. Dipl. -Kfm. Jörg Dahlkemper

Eingereicht am: 27. Januar 2022

Malte Dittmar

Thema der Arbeit

Kollisionsvermeidung eines mobilen Roboters mittels Schätzung der Bewegungsrichtung und Geschwindigkeit von Objekten aus 3D-Bilddaten

Stichworte

Kollisionsvermeidung, mobiler Roboter, (u,v)-Disparity Map, Kalman Filter, Stereokamera

Kurzzusammenfassung

Diese Arbeit befasst sich mit der Entwicklung eines Systems zur Kollisionsvermeidung, welches 3D-Bilddaten einer Stereokamera verwendet. Das System wird für einen mobilen Roboter mit dem Einsatzgebiet eines öffentlichen Parks entwickelt. Zur Lösung der Kollisionserkennung wird der Fokus auf recheneffiziente Algorithmen gelegt.

Malte Dittmar

Title of Thesis

Collision avoidance of a mobile robot using estimation of object motion direction and velocity from 3D image data.

Keywords

Collision avoidance, mobile robot, (u,v)-disparity map, Kalman filter, stereo camera.

Abstract

This thesis deals with the development of a collision avoidance system using 3D image data from a stereo camera. The system is developed for a mobile robot with the application area of a public park. The focus is on computationally efficient algorithms to solve the collision detection problem.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

CNN Convolutional Neural Network

UAV Unbemanntes Luftfahrzeug

ROS Robot Operating System

PWM Pulsweitenmodulation

ESA European Space Agency

USV Unbemanntes Wasserfahrzeug

GPS General Positioning System

Lidar Light detection and ranging Sensor

DSGVO Datenschutzgrundverordnung

UML Unified Modeling Language

FOV Field of View

MOT Multi Object Tracker

SORT Simple Online and Realtime Tracking

CPU Central Processing Unit

ROI Region of Interest

RCL ROS Client Library

ZRM Zustandsraummodell

MGD Mean Ground Depth

1 Motivation und Zielsetzung

1.1 Motivation

Autonome mobile Roboter finden in einem stetig wachsenden Einsatzgebiet Verwendung. Bekannte Beispiele sind Staubsaugerroboter, Rasenmäherroboter, unbemannte Zugmaschinen in der Landwirtschaft und natürlich das autonome PKW. In einem Forschungsprojekt der Stadt Hamburg in Kooperation mit der HAW Hamburg wird die autonome Müllbeseitigung in öffentlichen Parks aufgegriffen. Der Müll soll mit unbemannten Luftfahrzeugen (UAV) detektiert und die Positionen an autonome mobile Roboter gesendet werden. Die Roboter sollen eigenständig die Positionen anfahren und an den übermittelten Positionen Müll aufsammeln. Eine Herausforderung, die sich aus diesem Projekt ableiten lässt, ist die kollisionsfreie Fortbewegung in öffentlichen Parks mit mobilen Robotern.
Die Fortbewegung von autonomen mobilen Robotern im Straßenverkehr und in Innenräumen wird in verschiedenen Forschungsarbeiten aufgegriffen und die Forschung in diesen Bereichen ist schon weit fortgeschritten. Die Umgebung eines öffentlichen Parks hingegen ist nicht so stark verbreitet, wodurch in diesem Bereich interessante Erkenntnisse erlangt werden können.

1.2 Zielsetzung

In dieser Arbeit soll eine Kollisionsvermeidung entwickelt werden, welche es mobilen Robotern ermöglichen soll, sich kollisionsfrei zu bewegen. Die Umgebung des Roboters soll ein öffentlicher Park sein. Eine ausführliche Definition eines öffentlichen Parks ist in Kapitel 4.1 gegeben. Im Straßenverkehr hat die Sicherheit der Verkehrsteilnehmer die höchste Priorität. Um in komplexen Szenarien und bei hohen Geschwindigkeiten diese Sicherheit gewährleisten zu können, kommen verschiedene kostenintensive Sensorpakete

zum Einsatz. In dieser Arbeit bewegt sich der Roboter langsam und ist relativ leicht, wodurch er keinen großen Schaden verursachen kann. Mit diesem Hintergrund soll versucht werden, ein System zu entwickeln, welches kostengünstig ist und so robust ist, eine Kollisionsvorhersage sicher durchzuführen. Der dabei verwendete Ansatz soll einen Fokus auf recheneffiziente Verfahren legen, um das System in Robotern einsetzen zu können, die über eine geringe Rechenleistung verfügen.

2 Grundlagen

In diesem Kapitel werden die Grundlagen eingeführt, auf denen die in dieser Arbeit angewendeten Verfahren beruhen oder diese beinhalten. Die Theorie wird so genau erläutert, um ein Grundverständnis für die angewendeten Methoden zu erlangen.

2.1 Kameramodell

Dieser Abschnitt beschäftigt sich mit dem Modell einer Kamera. In dem ersten Teilabschnitt werden die Koordinatensysteme zum beschreiben von Positionen vorgestellt. In dem zweiten Teilabschnitt wird ein Kameramodell vorgestellt, mit dem die Abbildung eines 3D-Punktes beschrieben werden kann. Anschließend wird die Umrechnung der verschiedenen Koordinatensysteme beschrieben.

2.1.1 Raumkoordinaten

Es ist möglich, die Position P eines Objektes oder Punktes im Raum mit verschiedenen Koordinatensystemen zu beschreiben. In diesem Abschnitt werden die für diese Arbeit relevanten Koordinatensysteme vorgestellt. Das Roboterkoordinatensystem und das Kamerakoordinatensystem beschreiben die Position im Raum mit den kartesischen Koordinaten $P(x, y, z)$. Die Koordinatensysteme unterscheiden sich in ihrem Bezugspunkt und der Orientierung. Sie sind in Abbildung 2.1 dargestellt. Die verwendeten Indizes geben , um welches Koordinatensystem es sich handelt. Das Roboterkoordinatensystem beschreibt P über $P_r(x_r, y_r, z_r)$ und ist fest mit dem Roboter verankert. Die z_r-Koordinate gibt an, wie weit der Punkt von dem Roboter entfernt ist. Die x_r-Koordinate gibt an, wo sich der Punkt seitlich orientiert befinden. Die y_r-Koordinate gibt an, in welcher Höhe sich der Punkt befindet. Dieses Koordinatensystem kann dazu verwendet werden, die Position von Objekten direkt in Relation zum Roboter zu setzen. Es kann beispielsweise bei der Kollisionsvermeidung verwendet werden.

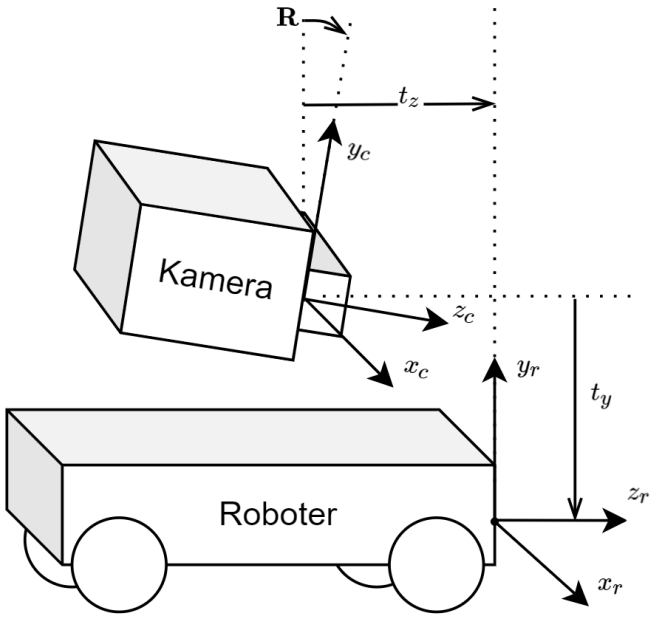


Abbildung 2.1: Zusammenhang zwischen Roboter und Kamerakoordinatensystem

Das Kamerakoordinatensystem ist mit der Kamera verankert. Die Position lässt sich mit $P_c(x_c, y_c, z_c)$ beschreiben. Die Koordinaten (x_c, y_c) spannen die Kameraebene auf, welche parallel zur Bildebene verläuft. Die z_c-Achse beschreibt die optische Achse, also die Entfernung eines Objekts zur Kameraebene.
Ein weiteres wichtiges Koordinatensystem ist das Bildkoordinatensystem, welches in Abbildung 2.2 dargestellt ist. Es beschreibt die Position eines 2D-Bildpunktes, welcher aus

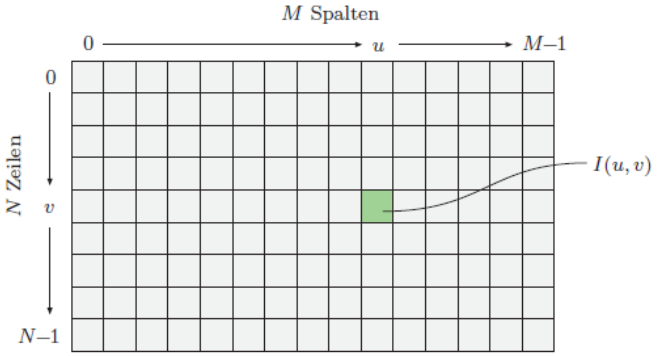


Abbildung 2.2: Bildkoordinatensystem [6]

dem 3D-Raum auf die Bildebene projiziert wird. Die Position des Bildpunktes P_b wird in Pixeln mit $P_b = I(u, v)$ beschrieben. Die u-Koordinate beschreibt die Position der

Spalte des aufgenommenen Pixels und die v-Koordinate die Pixel-Zeile. Die Werte für die Pixel werden in einer $(N \times M)$ großen Matrix I abgespeichert, wobei M die Anzahl der Spalten und N die Anzahl der Zeilen repräsentiert. Abhängig von der Auflösung der Kamera variiert die Größe von I. Bei Monochromkameras repräsentiert der Wert an der Position (u, v) die Intensität der Lichteinstrahlung. Handelt es sich um eine Farbkamera, so wird das Koordinatensystem um eine Achse erweitert, in welcher die Helligkeitswerte für die einzelnen Farbkanäle abgespeichert werden. Bei Stereokameras wird anstelle der Helligkeitswerte die Tiefeninformation des Bildpunktes hinterlegt. Bei den Bildkoordinaten ist zu beachten, dass es sich um ein linkshändiges Koordinatensystem handelt, bei dem die v-Achse von oben nach unten orientiert ist.

2.1.2 Lochkameramodell

Die Einführung des Lochkameramodells wird nach [3] vorgenommen. Um die Funktion einer Kamera möglichst einfach zu beschreiben, wird häufig das einfachste Modell einer Kamera, das Lochkameramodell, verwendet. Die idealisierte Lochkamera besteht aus einem geschlossenen lichtundurchlässigen Gehäuse, einer Blende mit einem infinitesimal kleinen Loch, durch das genau ein Lichtstrahl pro Bildpunkt einfällt, und einer Bildebene, auf der das Lichtbündel abgebildet wird. Die Bildebene ist parallel zur Blende der Lochkamera mit dem Abstand, der Bildweite b, angeordnet. Die Abbildung eines Objektpunktes auf die Bildebene lässt sich mit der Zentralprojektion

$$\begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} \mapsto \begin{pmatrix} x \\ y \end{pmatrix} = -\frac{b}{z_c} \begin{pmatrix} x_c \\ y_c \end{pmatrix} \tag{2.1}$$

beschreiben.
Das sich aus der Zentralprojektion ergebende (x, y)-Koordinatensystem wird im Folgenden als Bildkoordinatensystem bezeichnet. Werden die Koordinaten in Pixeln angegeben, werden die in Teilabschnitt 2.1.1 vorgestellten Koordinaten (u, v) verwendet, um die Position in (Zeilen, Spalten) anzugeben.
Um die Zentralprojektion in eine Matrixoperation umzuwandeln, werden die kartesischen Koordinaten in homogene Koordinaten transformiert, indem die Koordinatensysteme um eine zusätzliche Koordinate w erweitert werden. Die Rücktransformation wird durch die Division von w umgesetzt, weshalb $w \neq 0$ gewählt werden muss. Für die Kamerakoordi-

naten (x_c, y_c, z_c) und die Bildkoordinaten (u, v) ergeben sich die homogenen Koordinaten zu (x'_c, y'_c, z'_c, w_c) und (u', v', w). Die Zentralprojektion kann nun in der Matrixmultiplikation

$$\begin{pmatrix} u' \\ v' \\ w \end{pmatrix} = \begin{pmatrix} -b & 0 & 0 & 0 \\ 0 & -b & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x'_c \\ y'_c \\ z'_c \\ w_c \end{pmatrix} \tag{2.2}$$

beschrieben werden.

Um das Kameramodell zu vervollständigen, muss der Bildsensor bei der Modellierung berücksichtigt werden. Die Maße der einzelnen Pixel werden mit (s_x, s_y) für die x, y-Koordinaten angegeben. Um den Koordinatenursprung des Bildkoordinatensystems auf die optische Achse der Kamera zu legen, muss dieser um eine Anzahl von Pixeln (o_x, o_y) in x, y-Richtung verschoben werden. Die Anzahl der Pixel, um die der Koordinatenursprung verschoben werden muss, orientiert sich an der Auflösung des Bildes $(N \times M)$. Der Kamera-Koordinatenursprung liegt in der Mitte des Bildes. Ausgehend von einer geraden Anzahl an Pixeln bedeutet das für $P_c = (0,0)$, dass P_b zwischen $(\frac{M}{2}, \frac{N}{2})$ und $(\frac{M}{2}+1, \frac{N}{2}+1)$ liegt. Für die Berechnung der Verschiebung ergibt sich in x-Richtung

$$o_x = -\frac{M}{2} \tag{2.3}$$

und in y-Richtung

$$o_y = -\frac{N}{2}. \tag{2.4}$$

Für das Kameramodell ergibt sich unter der Berücksichtigung der Verschiebung des Koordinatenursprungs

$$\begin{pmatrix} u \\ v \end{pmatrix} = -\frac{b}{z_c} \begin{pmatrix} \frac{x_c}{s_x} \\ \frac{y_c}{s_y} \end{pmatrix} + \begin{pmatrix} o_x \\ o_y \end{pmatrix} \tag{2.5}$$

und für die Matrixmultiplikation in homogenen Koordinaten ergibt sich

$$\begin{pmatrix} u' \\ v' \\ w \end{pmatrix} = \begin{pmatrix} -\frac{b}{s_x} & 0 & o_x & 0 \\ 0 & -\frac{b}{s_y} & o_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x'_c \\ y'_c \\ z'_c \\ w_c \end{pmatrix}. \tag{2.6}$$

Aus (2.2) geht hervor, dass die Zentralprojektion nicht umkehrbar ist. Folglich ist es nicht möglich, anhand der Bildkoordinaten die genaue Position des Objektpunktes im

Raum zu bestimmen. Um dieser Herausforderung zu begegnen, können Stereokameras verwendet werden.

2.1.3 Koordinatentransformation

Wie auch schon das Lochkameramodell erfolgt die Koordinatentransformation nach [3] und wird für die Anwendung dieser Arbeit angepasst.
Das Roboter- und Kamerakoordinatensystem lassen sich jeweils durch eine Rotation und Translation ineinander umrechnen. Die Operation wird mit der Rotationsmatrix $\mathbf{R}$ und dem Translationsvektor $\mathbf{t}$ durchgeführt. Das Kamerakoordinatensystem ergibt sich zu

$$\begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} = \mathbf{R} \begin{pmatrix} x_r \\ y_r \\ z_r \end{pmatrix} + \mathbf{t}. \tag{2.7}$$

Bei der Koordinatentransformation werden, wie auch schon bei der Zentralprojektion, die homogenen Koordinaten verwendet, um die Rotation und die Translation in einer gemeinsamen Matrixmultiplikation auszuführen.

$$\begin{pmatrix} x'_c \\ y'_c \\ z'_c \\ w_c \end{pmatrix} = \begin{pmatrix} & \mathbf{R} & & \mathbf{t} \\ & & & \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x'_r \\ y'_r \\ z'_r \\ w_r \end{pmatrix} \tag{2.8}$$

2.2 Stereokamera

Aus Abschnitt 2.1.2 geht hervor, dass mit der Zentralprojektion die Tiefeninformationen eines abgebildeten Objektpunktes verloren gehen. Um die Entfernung eines beliebigen Objektpunktes berechnen zu können, werden unter anderem Stereokameras eingesetzt. Bei Stereokameras werden die Bilder von zwei Monokameras miteinander verrechnet, um die Tiefeninformationen zu erhalten.

2.2.1 Funktionsweise

Die Funktionsweise von Stereoaufnahmen wird am Beispiel von zwei komplanar angeordneten identischen Monokameras nach [3] erläutert. In Abbildung 2.3 ist der prinzipielle Aufbau dargestellt. Die Monokameras werden mit einer festen Entfernung a und einer

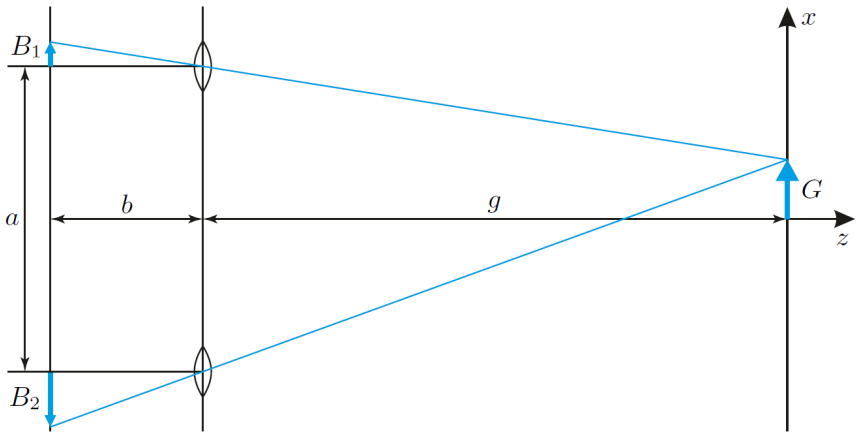


Abbildung 2.3: Modell einer Stereokamera [3]

identischen und konstanten Bildweite b nebeneinander platziert. Sie sind folglich auf der x_c-Achse zueinander verschoben. Sind die Bildebenen der Kameras parallel zueinander angeordnet, sind die Kameras komplanar. Die Bildebenen müssen nicht zwingend parallel angeordnet sein, um die Tiefeninformationen aus den aufgenommen Bildern berechnen zu können, in dieser Arbeit wird im weiteren Verlauf jedoch von einer komplanaren Anordnung ausgegangen. Bei einer komplanaren Anordnung verlaufen die optischen Achsen der beiden Kameras parallel zueinander. Das hat zur Folge, dass ein Objektpunkt, der mit beiden Kameras abgebildet werden kann, zum einen auf der gleichen y-Position abgebildet wird und zum anderen die gleiche Gegenstandsweite g aufweist, welche mit der Entfernung z_c gleichzusetzen ist. Bei der Abbildung unterscheidet sich folglich nur die x-Position. Abhängig von der Position auf der x-Achse wird der Objektpunkt P auf eine andere Position in der Bildebene b_1 und b_2 abgebildet. Die Achse, auf der die Verschiebung stattfindet, bei der komplanaren Anordnung die x-Achse, bezeichnet man Epipolarlinie. Die Herausforderung besteht darin, für jeden Bildpunkt aus der einen Kamera den korrespondierenden Bildpunkt in der anderen Kamera zu finden. Diese Herausforderung wird als Korrespondenzproblem bezeichnet und kann mit verschiedenen Stereo Matching Verfahren gelöst werden. Da sich die korrespondierenden Bildpunkte der beiden Kameras auf

der Epipolarlinie befinden, beschränkt sich das Stereo Matching auf den eindimensionalen Raum. Um Übereinstimmungen zu finden, werden die Bilder gefiltert und dadurch Merkmale extrahiert, anhand derer Übereinstimmungen gefunden werden können.
Die Abbildung auf den Bildebenen der Kameras wird mit B_1 und B_2, dem Abstand zu dem jeweiligen Kamerazentrum, bezeichnet. Die Verschiebung der Abbildung auf der Epipolarlinie wird Parallaxe oder auch Disparität p genannt und mit

$$p = B_1 - B_2 \tag{2.9}$$

berechnet. Mit der berechneten Disparität und den Konstanten a und b lässt sich die Tiefeninformation für jedes Pixel vollständig mit

$$\frac{g}{b} = \frac{a}{p} \tag{2.10}$$

beschreiben. Umgeformt nach der Gegenstandsweite g, lässt sich die Entfernung mit

$$z_c = g = \frac{a \cdot b}{p} \tag{2.11}$$

berechnen. Entfernt sich ein Objekt von der Bildebene werden die Bildpunkte näher an den Kamerazentren abgebildet, wodurch die Disparität kleiner wird.
Mit der gewonnenen Tiefeninformation lassen sich auch die fehlenden Kamerakooridinaten

$$x_c = \frac{a \cdot u}{p} \tag{2.12}$$

und

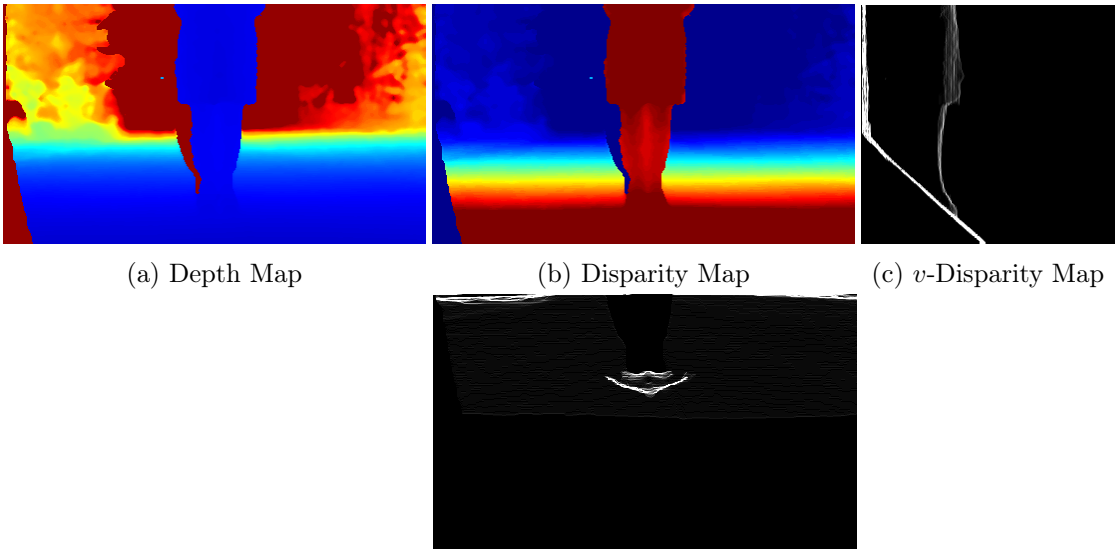
$$y_c = \frac{a \cdot v}{p} \tag{2.13}$$

beschreiben [29].

2.2.2 Darstellung der Tiefeninformationen

Die mit der Stereokamera gewonnenen Tiefeninformationen können auf verschiedene Arten dargestellt werden. In dieser Arbeit wird zum einen auf die sogenannte Disparity Map und zum anderen auf die Depth Map eingegangen. Die Disparity Map C_{disp} bildet sich aus den Disparitätswerten p und der jeweiligen Pixelposition u, v zu $C_{disp}(u, v, p)$ [28]. Entsprechend der Disparity Map lässt sich auch die Depth Map C_{depth} über die Pixelposition u, v, jedoch mit der Entfernung z_c über $C_{depth}(u, v, z_c)$ beschreiben. Die Depth Map

ist in Abbildung 2.4a und die Disparity Map in Abbildung 2.4b farblich dargestellt, um den Verlauf der Tiefeninformationen anschaulicher zu gestalten. Der Farbverlauf erfolgt von kleinen Werten in blau zu großen Werten in rot. Neben der Depth- und Disparity



(a) Depth Map (b) Disparity Map (c) v-Disparity Map

(d) u-Disparity Map

Abbildung 2.4: Darstellungsformen der Tiefeninformationen

Map sind die v-Disparity Map in Abbildung 2.4c und die u-Disparity Map in Abbildung 2.4d dargestellt. Sie werden aus der Disparity Map berechnet und weisen Eigenschaften auf, die bei dem Erkennen von Hindernissen nützlich sind.
Die v-Disparity Map wird aus Histogrammen von jeder Zeile der Disparity Map gebildet, welche ebenfalls zeilenweise in die v-Disparity Map eingetragen sind. Für eine v-Disparity Map des Datentyps uint8 wird der Wertebereich von 0-255 festgelegt und entlang der v-Achse die Werte eingetragen. Für eine Disparity Map der Auflösung $(N \times M)$ ergibt sich die Auflösung der v-Disparity Map zu $(N \times 256)$. Um die zeilenweise Abbildung gut sichtbar zu machen, ist die v-Disparity Map neben der Disparity Map abgebildet.
Die Abbildungen aus 2.4 zeigen einen ebenen Boden mit einer Person im Zentrum und Sträuchern im Hintergrund. In der v-Disparity Map verläuft ein gerader Strich von der unteren Mitte des Bildes schräg nach links oben. Diese Linie stellt den Boden dar. Die Linie geht in der Zeile in einen senkrechten Strich über, in dem die Entfernung des Bodens größer wird als der Arbeitsbereich der Kamera. Die Person, die sich im Bild befindet wird als annähernd senkrechter Strich wahrgenommen, da die Disparitäten für jede Zeile annähernd gleich sind. Eine wichtige Eigenschaft dieser Darstellungen ist, dass alles was

sich oberhalb des schräg durchs Bild verlaufenden Striches befindet als Objekt interpretiert werden kann.
Die u-Disparity Map wird nach einem ähnlichen Prinzip aufgebaut. Wie die Namen schließen lassen wird die u-Disparity Map ebenfalls aus Histogrammen gebildet. Diese werden jedoch spaltenweise, also entlang der u-Koordinate, erzeugt. Für das Beispiel einer Disparity Map mit einer $(N \times M)$ Auflösung ergibt sich für die u-Disparity Map die Auflösung mit dem Datentyp uint8 zu $(256 \times M)$. Um die spaltenweise Abbildung hervorzuheben, ist die u-Disparity Map unter der Disparity Map dargestellt
In der u-Disparity Map ist anders als in der v-Disparity Map der Boden nicht zu erkennen, was darin begründet ist, dass die Disparitätswerte in dem Bereich des Bodens konstant abnehmen, wodurch keine Häufungen für den Boden auftreten. Objekte, wie die Person in der Mitte hingegen, weisen abhängig von der Form in der Regel Häufungen für Disparitäten auf, wodurch diese als weiße Bereiche zu erkennen sind. Diese Eigenschaft kann dazu genutzt werden, um aus diesen Häufungen Objekte zu erkennen.

2.3 Kalman-Filter

Das im Jahr 1960 von R. E. Kalman entwickelte und nach ihm benannte Kalman-Filter [19] ermöglicht es aus verrauschten Messungen Zustände und Parameter eines linearen Systems zu schätzen [21]. Die Schätzung der einzelnen Zustände basiert auf einem physikalischen Modell des linearen Systems, welches in Zustandsraumdarstellung vorliegt.
In der Zustandsraumdarstellung werden die folgenden Bezeichnungen eingeführt, um das Modell zu beschreiben.

- $\mathbf{x}(k)$ ist der Zustandsvektor, welcher die Systemzustände enthält.
- $\mathbf{x}(k+1)$ ist der Zustandsvektor des folgenden Zeitschritts.
- $\mathbf{y}(k)$ ist der Ausgangsvektor. Er beschreibt die am System messbaren Ausgangsgrößen.
- $\mathbf{u}(k)$ ist der Eingangsvektor. Er beschreibt die Eingangsgrößen, die das System anregen.
- $\mathbf{A}$ ist die Systemmatrix. Sie beschreibt die interne Kopplung der Zustände, auch interne Dynamik genannt.

- **B** ist die Eingangsmatrix. Sie beschreibt den Zusammenhang des Eingangs auf die Zustände.
- **C** ist die Ausgangsmatrix. Sie beschreibt den Zusammenhang der Zustände auf Ausgangsgrößen.
- **D** ist die Durchgriffsmatrix. Sie beschreibt die direkten Auswirkungen der Eingangsgrößen auf die Ausgangsgrößen.

Das Zustandsraummodell verwendet in der diskreten Form die Gleichungen

$$\mathbf{x}(k+1) = \mathbf{A} \cdot \mathbf{x}(k) + \mathbf{B} \cdot \mathbf{u}(k) \tag{2.14}$$

$$\mathbf{y}(k) = \mathbf{C} \cdot \mathbf{x}(k) + \mathbf{D} \cdot \mathbf{u}(k) \tag{2.15}$$

zur Berechnung der Folgezustände $\mathbf{x}(k+1)$ und der Ausgangsgrößen $\mathbf{y}(k)$.
Die Schätzung der Zustände mit dem Kalman-Filter erfolgt in zwei Schritten. Der erste Schritt prädiziert die Folgezustände mit der Zustandsgleichung aus (2.14) und berechnet die Kovarianz des Schätzfehlers **P** aus vorangegangenen Zeitschritten. Der zweite Schritt führt die Korrektur der Schätzung und der Messung aus. Die Ergebnisse der Prädiktion werden in dieser Arbeit stets mit einem Dach und die Ergebnisse der Korrektur mit einer Tilde gekennzeichnet. Mit (2.16) werden die prädizierten Folgezustände und mit (2.17) die prädizierte Kovarianz des Schätzfehlers **P**, auch Unsicherheit genannt, berechnet. Die Prädiktion wird mit den Gleichungen

$$\hat{\mathbf{x}}(k+1) = \mathbf{A} \cdot \tilde{\mathbf{x}}(k) + \mathbf{B} \cdot \mathbf{u}(k) \tag{2.16}$$

$$\hat{\mathbf{P}}(k+1) = \mathbf{A} \cdot \tilde{\mathbf{P}}(k) \cdot \mathbf{A}^T + \mathbf{Q}(k) \tag{2.17}$$

durchgeführt. Das Prozessrauschen, welches durch die Ungenauigkeit bei der Modellierung hervorgerufen wird, wird mit der Matrix **Q** beschrieben.
Bei der Korrektur wird zuerst die Verstärkung **K** des Kalman-Filters berechnet. Sie hängt von der Unsicherheit der Modellierung und der Kovarianz des Messrauschens **R** ab. Mit der Verstärkung und den gemessenen Ausgangssignalen werden in (2.19) die geschätzten Zustände und in (2.20) die geschätzte Unsicherheit des Modells korrigiert. Damit der Kalman-Filter ideal arbeiten kann, muss das System- und Messrauschen mittelwertfrei und normalverteilt, die Fehlerquellen stochastisch unabhängig und das System linear

sein.

$$\mathbf{K}(k) = \hat{\mathbf{P}}(k) \cdot \mathbf{C}^T \cdot (\mathbf{C} \cdot \hat{\mathbf{P}}(\mathbf{k}) \cdot \mathbf{C}^T + \mathbf{R}(k))^{-1} \tag{2.18}$$

$$\tilde{\mathbf{x}}(k) = \hat{\mathbf{x}}(k) + \mathbf{K}(k) \cdot (\mathbf{y}(k) - \mathbf{C} \cdot \hat{\mathbf{x}}(k) - \mathbf{D} \cdot \mathbf{u}(k)) \tag{2.19}$$

$$\tilde{\mathbf{P}}(k) = (\mathbf{I} - \mathbf{K}(k) \cdot \mathbf{C}) \cdot \hat{\mathbf{P}}(k) \tag{2.20}$$

2.4 Schnittpunkt zweier Geraden

Die Berechnung des Schnittpunktes zweier Geraden kann dazu verwendet werden, die Kollision zweier sich mit konstanter Geschwindigkeit bewegender Punkte zu berechnen. Im folgenden wird die Berechnung des Schnittpunktes zweier Geraden nach [5] erläutert. Die beiden Geraden G und G' lassen sich über die ungleichen Punkte P_1, P_2 und P'_1, P'_2 mit den Gleichungen

$$G = \{P_1 + t(P_2 - P_1); t \in \mathbb{R}\}, \qquad G' = \{P'_1 + t'(P'_2 - P'_1); t' \in \mathbb{R}\} \tag{2.21}$$

beschreiben. Die Geraden sind in Abbildung 2.5 dargestellt. Der Schnittpunkt S der

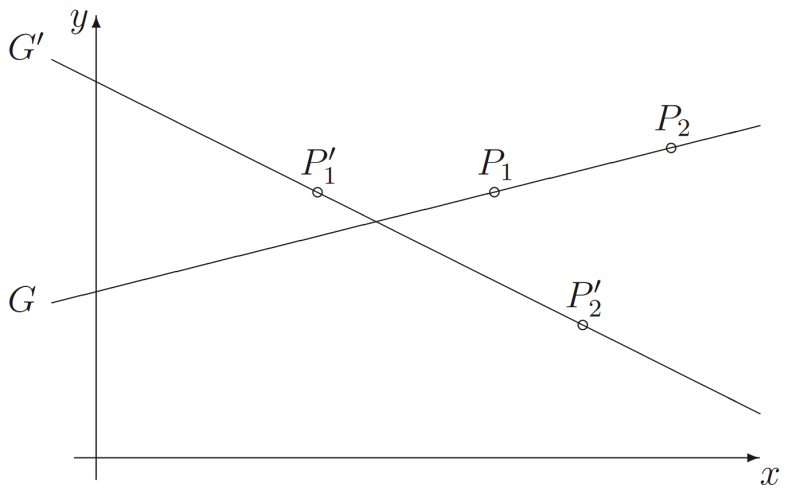


Abbildung 2.5: Schnittpunkt zweier Geraden [5]

beiden Geraden berechnet sich zu

$$P_1 + t(P_2 - P_1) = S = P'_1 + t'(P'_2 - P'_1). \tag{2.22}$$

Um den Schnittpunkt festlegen zu können, muss das Paar t und t' gefunden werden, welches die Gleichung

$$t(P_2 - P_1) + t'(P_1' - P_2') = P_1' - P_1 \tag{2.23}$$

erfüllt. In der Komponentenschreibweise lassen sich die Vektoren über

$$P_2 - P_1 = (\alpha_{11}, \alpha_{21}), \quad P_1' - P_2' = (\alpha_{12}, \alpha_{22}), \quad P_1' - P_1 = (\beta_1, \beta_2) \tag{2.24}$$

darstellen. Gelten die Bedingungen, dass $P_2 - P_1$ und $P_1' - P_2'$ linear unabhängig sind, $P_2 - P_1 \neq 0$ und $\alpha_{11} \neq 0$, dann lässt sich die Gleichung so umformen, dass sich für t und t'

$$t = \frac{\beta_1 \cdot \alpha_{22} - \beta_2 \cdot \alpha_{12}}{\alpha_{11} \cdot \alpha_{22} - \alpha_{21} \cdot \alpha_{12}} \quad , \quad t' = \frac{\beta_2 \cdot \alpha_{11} - \beta_1 \cdot \alpha_{21}}{\alpha_{11} \cdot \alpha_{22} - \alpha_{21} \cdot \alpha_{12}} \tag{2.25}$$

ergibt. Der Schnittpunkt lässt sich nach einsetzen von (2.25) in (2.23) über

$$S = P_1 + \frac{\beta_1 \cdot \alpha_{22} - \beta_2 \cdot \alpha_{12}}{\alpha_{11} \cdot \alpha_{22} - \alpha_{21} \cdot \alpha_{12}} \cdot (P_2 - P_1) = P_1' + \frac{\beta_2 \cdot \alpha_{11} - \beta_1 \cdot \alpha_{21}}{\alpha_{11} \cdot \alpha_{22} - \alpha_{21} \cdot \alpha_{12}} \cdot (P_2' - P_1') \tag{2.26}$$

berechnen. Anhand von t und t' lässt sich die Aussage treffen, ob der Schnittpunkt der beiden Geraden zwischen den Stützpunkten P_1, P_2 und P_1', P_2' liegt oder außerhalb. Für Werte von $0 \leq (t, t') \leq 1$ liegt der Schnittpunkt zwischen den Punkten und ansonsten außerhalb. Diese Information ist relevant, wenn es darum geht, den Schnittpunkt von Strecken zu berechnen.

2.5 ExoMy

Die Europäische Weltraumorganisation (ESA) setzt sich mit der ExoMars-Mission das Ziel, den Mars genauer zu erkunden. Es soll ein Rover auf den Mars geschickt werden, der dort Proben des Planeten entnimmt und diese zur Erde zurückbringt. Bei dem Rover handelt es sich um einen solarbetriebenen und von sechs Rändern angetriebenen Roboter (Abb. 6.1 rechts). Um das Thema für SchülerInnen und Studierende zugänglich zu machen und zu erreichen, dass sie sich vermehrt mit der Robotik auseinander setzen, wurde von der ESA ein Modell des Mars Rovers mit dem Namen ExoMy entwickelt.[10]
ExoMy ist ein unter der *GNU general public license* veröffentlichtes Open Source Projekt, welches den Nutzern ermöglicht, den Source Code frei zu verwenden [9]. ExoMy hat die Abmessungen Höhe $h = 42$ cm, Breite $b = 30$ cm und Länge $l = 40$ cm und ist in

Abbildung 6.1 links dargestellt. [10]

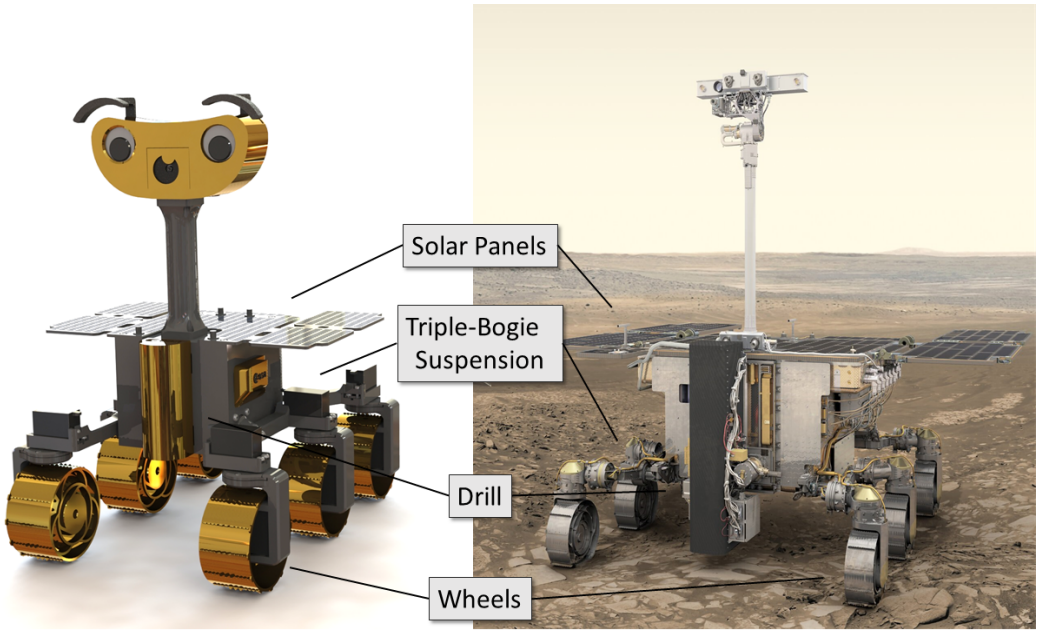


Abbildung 2.6: Vergleich ExoMy(links) und ExoMars Rover (rechts) [10]

ExoMy verfügt wie das Original über sechs-Antriebe, welche über einen Raspberry Pi 4 mit einem PWM Servo HAT angesteuert werden. Die Steuerung kann über einen USB-Controller oder über ein Web-Interface erfolgen. Mit einer Raspberry Pi-Kamera, welche im Kopf des Rovers verbaut ist, besteht die Möglichkeit, sich über das Web-Interface die aufgenommenen Bilder live anzuschauen und nach Ihnen zu navigieren.
Die Software von ExoMy verwendet das Roboter Framework ROS1, welches in Abschnitt 2.6 eingeführt wird. Es ist in Python umgesetzt und wird in Docker Containern ausgeführt.
Durch den sechs-Radantrieb und die flexiblen Achsen ist es mit dem Rover möglich durch unebenes Gelände zu fahren, wodurch er ideal in Outdoor Anwendungen eingesetzt werden kann. Durch die vielseitigen Einsatzmöglichkeiten und der Geländefähigkeit des Roboters, soll der ExoMy in dieser Arbeit die Grundlage für das zu entwickelnde System darstellen.

2.6 Roboterentwicklung mit ROS2

Bei der Entwicklung eines Roboters müssen unabhängig von dem Einsatzgebiet verschiedene Hardwarekomponenten zusammengeführt und miteinander abgestimmt werden, um einen reibungslosen Ablauf des vorgegebenen Verhaltens sicherzustellen. Des Weiteren muss ein Roboter mehrere Aufgaben bewältigen, welche die Expertise aus verschiedenen Fachrichtungen benötigt. Bei einem mobilen autonomen Roboter ist eine Kompetenz beispielsweise die Kollisionsvermeidung und eine andere die Regelung des Antriebs. Die Zusammenführung der einzelnen Systemkomponenten und die Verwaltung der Hardwarekomponenten ist meist mit großem Aufwand verbunden. Durch die Verwendung von ROS2 ist es möglich die einzelnen Komponenten separat von einander zu entwickeln und in der Umsetzung über vorab definiert Schnittstellen zusammenzuführen.
Im Jahr 2017 wurde erstmals eine vollwertige Version von ROS2, der Weiterentwicklung von ROS1 veröffentlicht. ROS2 bietet ROS1 gegenüber deutliche Verbesserungen in der Anwenderfreundlichkeit und wirkt insgesamt deutlich schlanker. ROS2 ist in C programmiert und bietet mit der ROS Client Library (RCL) eine Schnittstelle, die mit den offiziellen Bibliotheken *RCLPP* für C++ und *RCLPY* Python angesprochen werden kann. Es ist neben C++ und Python auch möglich die RCL-Schnittstelle mit anderen Programmiersprachen anzusprechen, die verfügbaren Bibliotheken sind jedoch nicht so vollständig wie RCLPP oder RCLPY.
Das Grundkonzept von ROS1 ist in ROS2 erhalten geblieben und es ist möglich, Projekte von ROS1 in ROS2 zu integrieren. [24]
ROS2 stellt eine Middleware bereit, die zwischen dem Betriebssystem und einzelnen ROS2 Anwendungen angesiedelt ist. In der ROS2 Middleware werden alle Anwendungen in dem sogenannten *ROS graph* eingetragen, welcher die Vernetzung der Nodes übernimmt. Die Kommunikation zwischen den Anwendungen wird mit Nodes umgesetzt. Der Vorgang ist in Abbildung 2.7 dargestellt.

Eine oder mehrere Nodes senden Messages an ein Topic und jede Node, die dieses Topic abonniert, empfängt diese Message. Das Senden einer Message an ein Topic wird mit einer Publisher-Instanz und das Empfangen einer Nachricht wird mit einer Subscription-Instanz umgesetzt. Die Kommunikation über Topics ist der Kommunikation von Queues sehr ähnlich. Der Unterschied liegt darin, dass es für ein Topic mehrere Sender und Empfänger geben kann und dass die Empfänger einen frei wählbaren Pufferspeicher haben, der angibt, wie viele Messages zwischengespeichert werden, bevor die älteste gelöscht wird. Die Struktur einer Message wird vorab festgelegt.

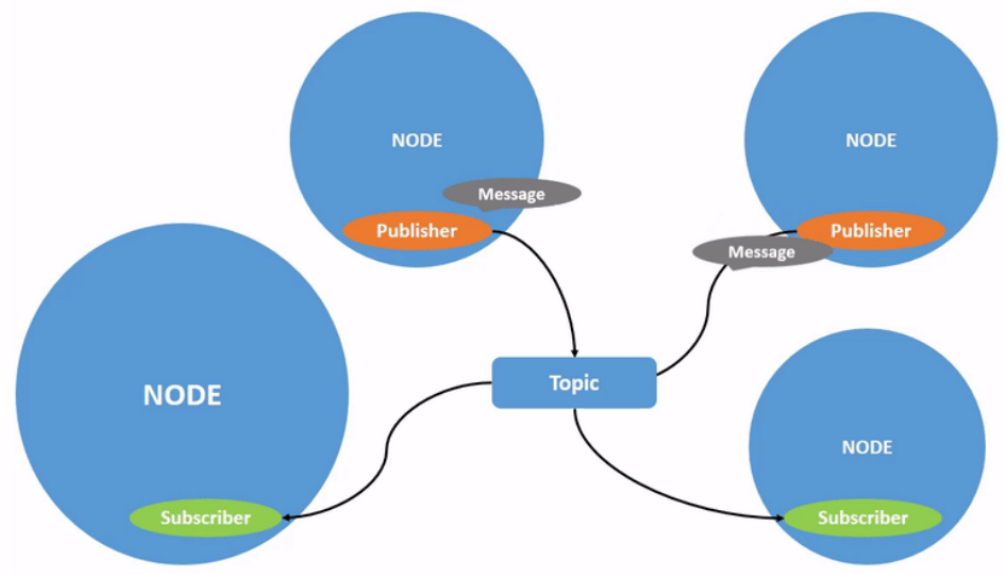


Abbildung 2.7: Senden von Messages über ROS-Topics [24]

Nodes und Messages können in separaten oder gemeinsamen Packages erzeugt werden. In den Packages werden die Abhängigkeiten zu anderen Packages definiert, was es einer Node erlaubt, beispielsweise Messages, Quellcode oder Definitionen von anderen Packages einzubinden.
Eine andere Art der Kommunikation ist in Abbildung 2.8 dargestellt. Die Kommunikation wird mit ROS-Services umgesetzt.

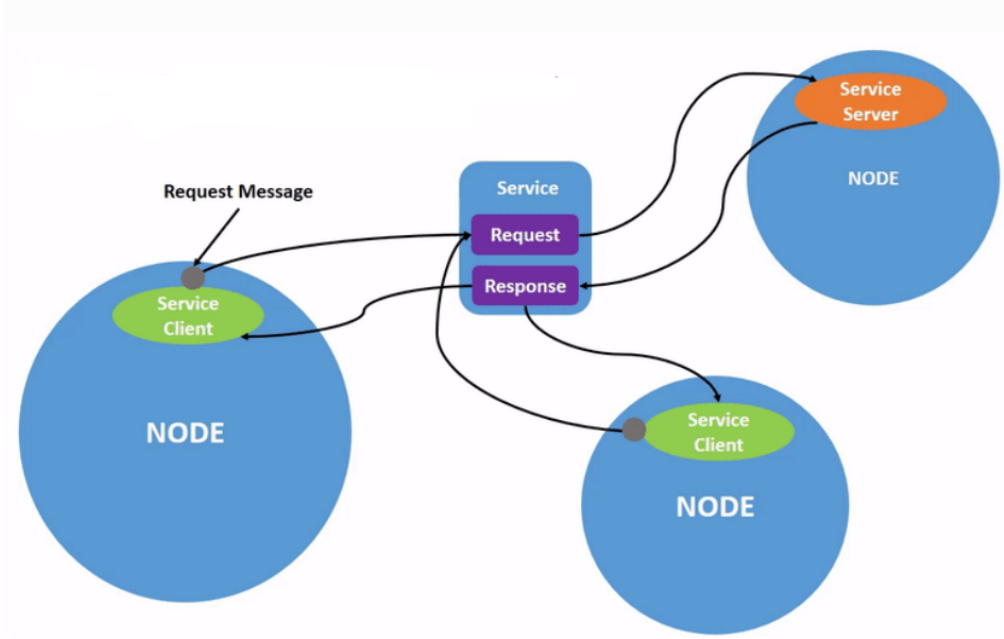


Abbildung 2.8: Senden von Messages über ROS-Services [24]

Services funktionieren nach dem Server-Client Prinzip. Der Server stellt einen Service bereit, der ausgeführt wird, wenn er von einem Client angefragt wird. Anders als bei der Kommunikation über Topics, kann es nur einen Server, jedoch mehrere Clients geben, die den Service abonnieren. Bei dem Service ist wie bei der Topic vorab festgelegt, welche Daten die Anfrage des Clients enthalten muss und welche Daten die Antwort des Servers enthält. Der Service sorgt zusätzlich dafür, dass nur der Client, der eine Anfrage gesendet hat, auch die dazugehörige Antwort bekommt. Die Kommunikation mit Services ist an den Punkten sinnvoll, wo eine Aufgabe einmalig ausgeführt werden muss, wie beispielsweise eine Pfadplanung.
Parameter werden verwendet, um eine Node zu konfigurieren. Sollten sich Parameter während der Laufzeit verändern, besteht die Möglichkeit die Parameter über Services anzupassen.

3 Stand der Technik

Dieses Kapitel stellt einen Ausschnitt von verschiedenen Methoden dar, die zum Zeitpunkt des Erstellens der Arbeit als Stand der Technik gelten. Die spätere Umsetzung der Arbeit orientiert sich an diesen Methoden.

3.1 Anwendungsbereiche und Sensoren in autonomen Fahrzeugen

Autonome mobile Roboter finden in immer mehr Bereichen Anwendung, sei es zum Transport von Gütern in der Industrie mit Flurförderfahrzeugen, beim Servieren von Speisen und Getränken in Restaurants oder im Automobil. In allen Bereichen ist eine Kollisionsvermeidung zwingend erforderlich, ist jedoch für die verschiedenen Anwendungsgebiete unterschiedlich komplex. Fahrzeuge, die sich im Freien bewegen, müssen in der Regel mehr Faktoren beachten, als Fahrzeuge die sich in einem Gebäude bewegen. Im Folgenden werden exemplarisch zwei Fahrzeuge vorgestellt, die das autonome Fahren im Automobil aufgreifen.

Automatisiertes Fahren im Automobil

In diesem Abschnitt werden zwei Beispiele für automatisiertes Fahren im Automobil vorgestellt. Bei dem automatisierten Fahren im Automobil wird der Automatisierungsgrad in Stufen eingeteilt. Eine Einteilung wurde durch das International On-Road Automated Vehicle Standards Committee der Society of Automotive Engineers im Report J3016 [26] vorgenommen. Die Einstufung erfolgt dabei in sechs Automatisierungsstufen.

- Stufe 0: Kein Unterstütztes Fahren
- Stufe 1: Fahrerassistenzsysteme

- Stufe 2: Teilweise automatisiertes Fahren
- Stufe 3: Bedingtes automatisiertes Fahren
- Stufe 4: Hoch automatisiertes Fahren
- Stufe 5: Vollautomatisiertes Fahren

Diese Unterteilung dient vielen Forschungsgruppen und Entwicklern als Anhaltspunkt zur Einstufung des Automatisierungsgrades [22]. Die in dieser Arbeit vorgestellten Systeme sind ebenfalls nach [26] eingestuft. Zum einen wird das von Daimler entwickelte DRIVE PILOT System, welches dem Fahrer das automatisierte Fahren der Stufe 3 ermöglicht [8], und zum anderen das Waymo Driver System, welches in Amerika bereits in fahrerlosen Taxen eingesetzt wird und das automatisierte Fahren der Stufe 5 erfüllt [18], vorgestellt.

DRIVE PILOT

In [8] wird das, in der Mercedes S Klasse verbaute Assistenzsystem DRIVE PILOT, welches das automatisierte Fahren der Stufe 3 ermöglicht, vorgestellt. Das automatisierte Fahren der Stufe 3 übernimmt beispielsweise in Staus auf Autobahnen die Kontrolle über das Auto und steuert es. Tritt eine Situation ein, die der DRIVE PILOT nicht beherrscht, wird der Fahrer aufgefordert die Kontrolle zu übernehmen. Um das automatisierte Fahren der Stufe 3 zu verwirklichen, sind rund um das Auto verschiedene Sensoren verbaut. In der Windschutzscheibe ist eine Stereokamera mit einem Öffnungswinkel von 70° verbaut. Im Kühlergrill befindet sich ein Fernbereichsradarsensor, der mit einem zusätzlichen *Light detection and ranging Sensor* (Lidar) kombiniert wird. Mit dem Radarsensor soll eine hohe Ausfallsicherheit bei schlechter Sicht gewährleistet und mit dem Lidar eine höhere Genauigkeit erzielt werden. Jeweils links und rechts befinden sich zusätzlich insgesamt vier Nahbereichsradarsensoren an den Stoßstangen, die mit einem Öffnungswinkel von 130° eine 360° Rundumsicht ermöglichen. Zusätzlich zu den Sensorinformationen erhält das Fahrzeug über eine Online-Karte genaue Informationen zu Straßeneigenschaften, Verkehrszeichen und Baustellen. Um die vielen Informationen auswerten zu können, befindet sich im Fahrzeug ein leistungsfähiges Zentralsteuergerät.

Waymo Driver System

Im März 2020 wurde die fünfte Generation des Waymo Driver Systems veröffentlicht [18]. Bei dem System handelt es sich um ein autonomes Fahrsystem, bestehend aus verschiedenen Sensoren, einem Hochleistungsrechner und der zugehörigen Software. Das System wird bereits in den USA im Straßenverkehr eingesetzt und laufend verbessert.
Das Sensorpaket des Waymo Driver besteht aus zwei Fernbereichsradarsensoren und einer Fernbereichskamera, welche in Fahrtrichtung oberhalb der Windschutzscheibe angeordnet sind. Zusätzlich verfügt das System über einen 360° Grad Lidar, der zusammen mit einer 360° Kamera auf dem Dach des Autos montiert ist. Bei dem Lidar handelt es sich um einen Sensor mit einer Reichweite von 300 m. Zusätzlich sind auf Höhe der beiden Vorderräder eine Kombination aus Lidar, Radar und einer Kamera installiert. Auf der Höhe des vorderen und hinteren Nummernschildes sind jeweils ein Weitwinkellidar und eine Weitwinkelkamera installiert. Auf der Höhe der hinteren Stoßstange sind zusätzlich auf jeder Seite ein Radarsensor und eine Kamera installiert. In Summe sind das acht Kameras, sechs Radarsensoren und fünf Lidar Sensoren, welche die Funktion des autonomen Fahrens sicherstellen sollen. Um die anfallenden Datenmengen verarbeiten zu können, arbeitet Waymo seit 2009 in Kooperation mit dem Chiphersteller Intel und entwickelt seine eigenen Hochleistungsrechner.
Um Kollisionen zu vermeiden, laufen neben dem Hauptsystem unabhängige Kollisionserkennungs- und Kollisionsvermeidungssysteme, welche das Fahrzeug jederzeit abbremsen oder stoppen können. Zudem sind viele Systeme redundant umgesetzt, um eine Falschberechnung oder einen Systemausfall abzufangen. [18]

3.2 Verwandte Arbeiten

In [13] wird ein System zur Hindernisvermeidung auf unbemannten Wasserfahrzeugen (USV) vorgestellt. Diese Arbeit verwendet eine Kombination aus einer Stereokamera zur Form- und Farberkennung und einem 3D-Lidar für eine allgemeine Objekterkennung. Für die Form- und Farberkennung wird in Echtzeit ein Tiny YOLOv3, Convolutional Neural Network (CNN), eingesetzt. Mit einer NVIDIA GEFORCE GTX 1080 wird in dieser Arbeit eine Bildwiederholungsrate von 39Hz erreicht. Die erkannten Objekte werden für die Orientierung bei der Navigation in eine Datenbank gespeichert. Die Objekterkennung des 3D-Lidar erfolgt mit einem Öffnungswinkel von 90°, das FOV wird in drei Bereiche eingeteilt. Wird in dem mittleren Bereich, der Fahrspur, ein Hindernis erkannt, wird geprüft,

ob links oder rechts ein Bereich frei ist und in diesem Bereich weiter gefahren. Dieser Ansatz funktioniert für den Einsatz von USV sehr gut, da die Anzahl der Hindernisse auf dem Wasser relativ gering ist.
Ein anderer Ansatz wird in [1] für die Kollisionsvermeidung eines unbemannten Flugobjektes (UAV) vorgestellt. Die Arbeit verwendet eine Monokamera, die Objekte anhand von Merkmalen erkennt. Der Algorithmus verwendet jeweils zwei Bilder, das erste Bild wird von der Kamera aufgenommen und mit dem zweiten Bild, welches in einer Datenbank liegt, abgeglichen. In der Datenbank liegen Bilder von allen zu erkennenden Objekten. Ein Objekt kann also nur erkannt werden, wenn es auch in der Datenbank liegt. Um ein Objekt zu erkennen, werden aus dem aufgenommenen Bild Merkmale nach dem SURF-Algorithmus extrahiert. Die Erkennung läuft in drei Schritten, Merkmale extrahieren, Merkmale beschreiben und Merkmale zusammenbringen.
Anders als in [1] verwendet der in [23] veröffentlichte Ansatz eine Stereokamera, mit der Objekte erkannt und verfolgt werden, um die Bewegungen der Objekte bei der Kollisionsvermeidung zu berücksichtigen. In der Arbeit wird die u-Disparity Map verwendet, um Objekte vom Hintergrund zu trennen und als Hindernisse zu detektieren. Mit dieser Methode ist es möglich, unbewegte Hindernisse zu erkennen und eine Kollision mit Ihnen zu vermeiden. In [20] wird das Verfahren zur Objekterkennung aufgegriffen und um eine Objektverfolgung erweitert, um die Bewegung der Objekte bei der Kollisionsvermeidung zu berücksichtigen. Die Zuordnung der Objekte in aufeinander folgenden Frames wird durch die Auswertung der Gaußschen Wahrscheinlichkeitsdichte vorgenommen. Die Vorhersage über den Bewegungszustand wird mit einem linearen Kalman-Filter umgesetzt.
In [27] wird ein Ansatz vorgestellt, bei dem ein Kollisionsvermeidungssystem für den Straßenverkehr entwickelt wird. Der Ansatz verwendet die v-Disparity Map dazu den Übergang von der Straße zum Himmel zu finden und die Bereiche zu trennen. Die Straße wird als Region of Interest (ROI) festgelegt und gibt den Bereich an, in dem Hindernisse erkannt werden müssen. Die u-Disparity Map wird dazu verwendet, Hindernisse in dem Sichtfeld (FOV) zu erkennen. Mit dem optischen Fluss, wird zum einen die Eigenbewegung des Fahrzeugs und zum anderen die Bewegung der erkannten Hindernisse ermittelt. Anhand der Position der Objekte und der berechneten Geschwindigkeiten wird die Wahrscheinlichkeit einer Kollision ermittelt. Anhand der Kamerabilder lassen sich weit entfernte Objekte nicht detektieren und es kann keine Kollision vorhergesagt werden. Aus diesem Grund wird ein Radarsensor verbaut, der die weit entfernten Objekte erkennt und bei der Kollisionsvermeidung berücksichtigt.

3.3 Objektverfolgung

Bei der Objektverfolgung (engl. object tracking) geht es darum, erkannte Objekte in einer Bildfolge zu verfolgen. Im Folgenden werden zwei Methoden vorgestellt, mit denen es möglich ist, Objekte zu verfolgen. Die Erste befasst sich mit dem optischen Fluss, eine Allgemeine Methode, die in verschiedenen Algorithmen eingesetzt wird. Die Zweite ist ein sogenannter Multi Object Tracker (MOT). MOT werden eingesetzt, um mehrere Objekte gleichzeitig zu verfolgen. Es gibt mehrere Datensätze, auf die verschiedene Tracker angewendet und die Ergebnisse verglichen werden. Einer dieser Datensätze ist der Kitti Datensatz [12]. Ein MOT, der sehr kurze Ausführungszeiten aufweist, ist der Simple Online an Realtime Tracking Algorithmus (SORT). Dieser wird als zweites vorgestellt.

3.3.1 Optischer Fluss

Verändert sich die aufgenommene Szene einer Kamera, beispielsweise durch sich bewegende Objekte oder die Bewegung der Kamera selbst, ist es in vielen Anwendungen relevant, diese Bewegungen nachverfolgen zu können. Eine weit verbreitete Methode ist die Berechnung des optischen Flusses. Der optische Fluss gibt das Vektorfeld einer Bildsequenz an, welches die zweidimensionale Bewegung jedes Bildpunktes beschreibt [15]. Um den optischen Fluss berechnen zu können, müssen in aufeinander folgenden Bildern geeignete Merkmale extrahiert und entsprechend dem nachfolgenden Bild zugeordnet werden. Die Zuordnung der gefundenen Merkmale trifft wie auch schon das Stereo Matching auf das Korrespondenzproblem, welches bereits in Abschnitt 2.2.1 erläutert wird. Beim optischen Fluss kommt jedoch erschwerend hinzu, dass es keine Epipolarlinie gibt, die das Problem auf den eindimensionalen Raum eingrenzt. Die Zuordnung muss im zweidimensionalen Raum stattfinden, wodurch mehr Rechenleistung benötigt wird. Beim optischen Fluss muss zwischen dem sogenannten dichten- (engl. dense optical flow) und dem spärlichen optischem Fluss (engl. sparse optical flow) unterschieden werden. Der dichte optische Fluss hat den Vorteil, dass es sehr einfach ist, Merkmale zu extrahieren. Der Nachteil liegt darin, dass es sehr aufwändig ist, viele Merkmale zuzuordnen. Der spärliche Fluss hat den Vorteil, dass es nur wenige Merkmale gibt, die zugeordnet werden müssen, der Nachteil hier besteht jedoch darin, dass es sehr aufwendig ist, stabile Merkmale zu extrahieren, die sicher zugeordnet werden können. Auf der Website des Kitti-Datensatzes werden verschiedene Algorithmen zur Berechnung des optischen Flusses miteinander verglichen. Ein Großteil der Algorithmen verwendet eine GPU oder mehrere Prozessoren,

um die Berechnungen möglichst schnell parallel ausführen zu können. Aufgrund der benötigten Rechenleistung, wird in dieser Arbeit davon abgesehen, den optischen Fluss für die Objektverfolgung zu verwendet.

3.3.2 Simple Online and Realtime Tracking

Mit dem Simple Online and Realtime Tracking-Algorithmus (SORT)-Algorithmus wurde im Jahr 2016 ein MOT veröffentlicht, der effiziente Algorithmen verwendet und damit neben Genauigkeit einen Fokus auf Geschwindigkeit legt [2]. Mit mathematischen Methoden wie einem Kalman-Filter und der Ungarischen Methode sollen Zykluszeiten von 260 Hz auf herkömmlichen PC erreicht werden können [2]. Auf der Website des Kitti-Datensatzes ist der Algorithmus sogar mit einer Ausführungszeit von $0,002$ s bei einer CPU mit einer $2,5$ GHz Taktung aufgeführt [11]. Im Folgenden wird der SORT-Algorithmus nach [2] näher erläuter.
Der SORT-Algorithmus verwendet ein Modell des Objektes, welches verfolgt werden soll. Bei dem Modell handelt es sich um das kleinstmögliche Rechteck, welches die Kontur des Objektes komplett umschließt, die sogenannte Bounding Box. Die Bounding Box lässt sich mit dem Parametervektor

$$\mathbf{x}_{bb} = \Big(u_{bb}, v_{bb}, w, h\Big) \tag{3.1}$$

beschreiben. Die Parameter u_{bb}, v_{bb} stehen für die Bildkoordinaten des unteren linken Punktes der Bounding Box und die Parameter w, h stehen für die Breite und Höhe der Bounding Box. Der SORT Algorithmus verwendet für die Objektverfolgung den Mittelpunkt u, v, die Fläche s und das Seitenverhältnis r der Bounding Box. Diese Werte müssen aus dem Parametervektor $\mathbf{x}_{bb}$ mit

$$u = u_{bb} + \frac{w}{2}, \qquad v = v_{bb} + \frac{h}{2}, \qquad s = w \cdot h, \qquad r = \frac{w}{h} \tag{3.2}$$

berechnet werden. Um die Objekte im nachfolgenden Bild wieder zu finden, wird für jedes erkannte Objekt ein separates Kalman-Filter erzeugt, mit dem der Folgezustand geschätzt wird. Dadurch wird die Wahrscheinlichkeit erhöht, dass die Objekte im Folgebild wieder gefunden werden. Für das Kalman-Filter werden die Parameter in den Zustandsraum überführt und ein Zustandsraummodell mit dem Zustandsvektor

$$\mathbf{x}_{sort} = \Big(u \quad v \quad s \quad r \quad \dot{u} \quad \dot{v} \quad \dot{s}\Big)^T \tag{3.3}$$

definiert. Das Zustandsraummodell beschränkt sich auf ein konstantes Geschwindigkeitsmodell, das bedeutet, der Folgezustand berechnet sich am Beispiel der u-Koordinate mit

$$u(k+1) = u(k) + \dot{u}(k). \tag{3.4}$$

Für das Zustandsraummodell des SORT-Algorithmus ergibt sich die Berechnung der Folgezustände über

$$\mathbf{x}_{sort}(k+1) = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u(k) \\ v(k) \\ s(k) \\ r(k) \\ \dot{u}(k) \\ \dot{v}(k) \\ \dot{s}(k) \end{pmatrix}. \tag{3.5}$$

Da die zeitlichen Änderungen nicht aus den einzelnen Bildern hervorgehen, ergibt sich die Ausgabegleichung zu

$$\mathbf{y}(k) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} u(k) \\ v(k) \\ s(k) \\ r(k) \\ \dot{u}(k) \\ \dot{v}(k) \\ \dot{s}(k) \end{pmatrix}. \tag{3.6}$$

Mit dem in(3.5) und (3.6) definierten Modell kann der SORT-Algorithmus Objekte nach dem Ablauf in Abbildung 3.1 verfolgen. Der Algorithmus liest zu Beginn eines Durchlaufs neue Objekte ein und berechnet die Zustandsvektoren. Aus dem vorherigen Durchlauf, können bereits Kalman-Filter instanziiert sein. Die Kalman-Filter führen einen Prädiktionsschritt aus, um die Zustände für das aktuelle Bild zu schätzen. Nach dem Prädiktionsschritt wird die Intersection over Unit (IOU), der Bounding Boxes von jedem Objekt mit jedem Kalman-Filter berechnet und in einer Matrix gespeichert. Anschließend werden die Objekte den Kalman-Filtern aufgrund der berechneten IOUs nach der Ungarischen Methode zugeordnet. Dabei werden nur Paare berücksichtigt, bei denen die IOU $> 0{,}3$ ist. Nach der Zuordnung führen die Kalman-Filter, für die ein Objekt gefunden wird, einen

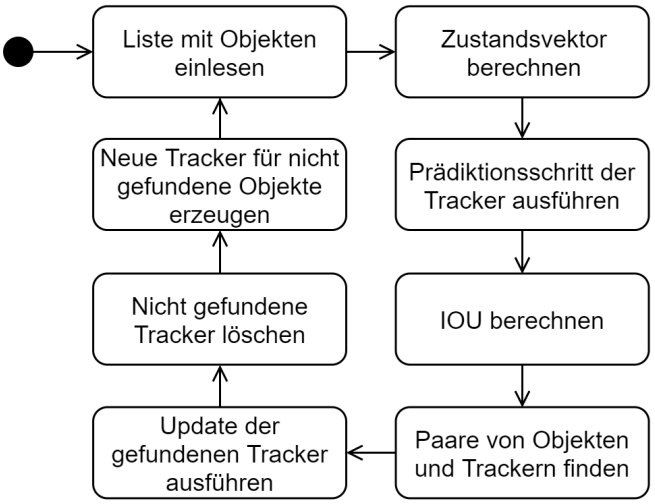


Abbildung 3.1: SORT-Ablaufdiagramm

Update-Schritt aus. Für den Updateschritt werden die Zustände des Objektes verwendet. Für Objekte, die keinem Kalman-Filter zugeordnet werden, werden neue Kalman-Filter instanziiert. Bei der Initialisierung wird davon ausgegangen, dass die Objekte sich nicht bewegen. Die Zustände $\dot{u}, \dot{v}, \dot{s}$ werden also mit 0 initialisiert. Die Kalman-Filter, denen kein Objekt zugeordnet werden kann, werden gelöscht.

4 Analyse der Anforderungen

Um die Anforderungen an das gesamte System definieren zu können, müssen verschiedene Aspekte berücksichtigt werden. Im ersten Abschnitt wird analysiert, welcher Systemkontext zu betrachten ist und welche Stakeholder es für das zu entwickelnde System gibt. Aus dem Systemkontext und unter Berücksichtigung der Stakeholder werden die verschiedenen Anwendungsfälle definiert, aus denen Anforderungen an das zu entwickelnde System abgeleitet werden. Anschließend werden die rechtlichen Rahmenbedingungen, in denen sich das System bewegen darf, ermittelt. Abschließend werden die aus den vorangegangenen Abschnitten hergeleiteten Anforderungen an das zu entwickelnde System zusammengefasst und aufgelistet.

4.1 Systemkontext

Der Systemkontext beinhaltet alle Teile der Umgebung, die einen relevanten Einfluss auf das System haben können. Im Systemkontext legt die Systemgrenze fest, welche Aspekte das zu entwickelnde System beinhaltet und welche Aspekte außerhalb des Systems liegen und über Schnittstellen mit dem System in Verbindung treten.[7]
Wie in Kapitel 1 eingeführt, soll in dieser Arbeit eine Möglichkeit gefunden werden, einen Roboter kollisionsfrei durch einen Park fahren zu lassen. Aus dieser Zielsetzung leitet sich die Aufgabenstellung zur Entwicklung eines Systems zur Kollisionsvermeidung ab.

Die Umgebung des Roboters

Um den Systemkontext aus dieser Vorgabe zu definieren, wird vorab festgelegt, was in dieser Arbeit unter einem Park zu verstehen ist und welche Eigenschaften dieser besitzt. Im Duden wird ein Park wie folgt definiert [4]: „größere [einer natürlichen Landschaft

ähnliche] Anlage mit [alten] Bäumen, Sträuchern, Rasenflächen, Wegen [und Blumenrabatten]“. Diese kurze Definition beschreibt, was in dieser Arbeit unter einem Park zu verstehen ist. Die Definition bringt folgende Eigenschaften mit sich.

- Der Untergrund unterscheidet sich zwischen Schotter- oder Sandwegen, gepflasterten Flächen oder Rasenflächen.
- Starre Objekte sind beispielsweise Bäume, Sträucher, Bänke, Mülleimer, stehende Personen oder Fahrräder.
- Sich bewegende Objekte sind beispielsweise Fußgänger, Hunde oder Fahrradfahrer.
- Neben diesen Objekten, gibt es Seen oder Blumenrabatten, die großflächig auftreten können oder von Sensoren schwierig als Hindernisse zu erkennen sind. Für diese Art von Objekten soll die Kollisionsvermeidung nicht ausgelegt sein. Diese Bereiche sollen in einer weiterführenden Arbeit als Sperrbereiche definiert werden, um die der Roboter herum navigiert wird.
- Weitere Sperrbereiche sind Fahrradwege oder Straßen, die ebenfalls nicht als Hindernisse zu erkennen sind.
- Eine weitere Eigenschaft sind sich verändernde Lichtverhältnisse durch Schatten von Wolken oder Bäumen. Regen ist auch eine Eigenschaft, die in Parks zu berücksichtigen ist, da der Roboter jedoch nicht wasserfest ist, wird davon ausgegangen, dass er nur bei Trockenheit eingesetzt wird.

Die Systemgrenze

Hardwareseitig bezieht sich das System auf den Roboter. Daraus folgt, dass sich die hardwareseitige Systemgrenze auf die Abmessungen des Roboters bezieht. Softwareseitig wird das System auf die Kollisionsvermeidung begrenzt. Da die Kollisionsvermeidung nur eines von mehreren Teilsystemen ist, aus denen sich das Gesamtsystem, der Roboter, zusammensetzt (Abb. 4.1), werden die anderen Teilsysteme als sogenannte Nachbarsysteme betrachtet. Ein Nachbarsystem ist beispielsweise die Steuerung des Roboters, welche über Schnittstellen mit der Kollisionsvermeidung kommuniziert. Weitere Teilsysteme sind die Kartierung der Umgebung oder die Pfadplanung zum Anfahren verschiedener Ziele. Diese Nachbarsysteme sind noch nicht umgesetzt, werden im Systemkontext jedoch mit

berücksichtigt, um das Design des Systems für solche Erweiterungen zugänglich zu gestalten.
Neben den Nachbarsystemen gehört die physische Umgebung des Roboters zum Systemkontext und befindet sich außerhalb der Systemgrenze. Hier müssen die Eigenschaften von öffentlichen Parks berücksichtigt werden, da sich das Einsatzgebiet des Roboters auf diese bezieht. Wichtige Aspekte sind sowohl starre, als auch sich bewegende Objekte, die eine Kollision hervorrufen können. Zu den starren Hindernissen gehören beispielsweise Bäume oder Mülleimer. Zu den Beweglichen gehören Spaziergänger oder Fahrradfahrer. Eine Veränderung der Umgebung, beispielsweise durch sich verändernde Lichtverhältnisse hervorgerufen, kann einen Einfluss auf das System nehmen und muss berücksichtigt werden. Zur Umgebung zählt auch der rechtliche Rahmen, in dem sich der Roboter bewegen darf. Ein wichtiger Aspekt für die Kollisionsvermeidung ist das Erfassen und Verarbeiten personenbezogener Daten.

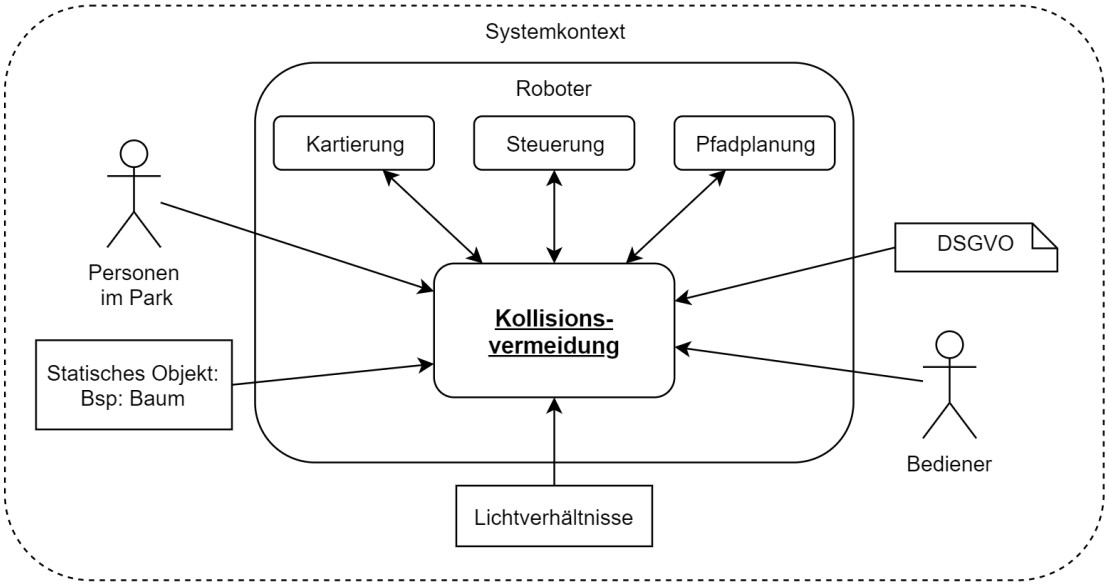


Abbildung 4.1: Systemkontext

4.2 Stakeholder

Bei der Entwicklung der Kollisionsvermeidung müssen Stakeholder, die verschiedene Anforderungen an das System stellen, berücksichtigt werden. Im Folgenden werden die Stakeholder und ihre Interessen analysiert.

Ein Teil der Stakeholder leitet sich direkt aus den Aktoren des Systemkontextes her. Der andere Teil setzt sich aus Personengruppen zusammen, die nicht im Systemkontext enthalten sind, aber ein Interesse an dem zu entwickelnden System haben. Die Stakeholder sind in Tabelle 4.1 aufgeführt. Aus dem Systemkontext leiten sich die Stakeholder Personen im Park, Bediener des Systems und die Datenschutzgrundverordnung (DSGVO) ab.

Personen im Park: Die Personen, die sich im Park befinden, haben ein Interesse daran, dass die Kollisionsvermeidung funktioniert und frühzeitig eine Warnmeldung absetzt, wenn eine Kollision bevorsteht oder der Abstand zum Roboter zu klein wird, und dass die DSGVO eingehalten wird.

Bediener des Roboters: Der Bediener hat ein Interesse daran, dass das System möglichst einfach in Betrieb genommen werden kann und möglichst fehlerfrei funktioniert.

DSGVO: Die DSGVO wird in dieser Arbeit ebenfalls als Stakeholder geführt, da durch ihre Einhaltung klare Anforderungen an das System gestellt werden. Das Interesse der DSGVO liegt in der rechtmäßigen Verarbeitung personenbezogener Daten.
Die Stakeholder, die nicht direkt aus dem Systemkontext hergeleitet werden können, sind der Betreiber öffentlicher Parks, der Ersteller dieser Arbeit, weiterführende Forschungsarbeiten, die auf dieser Arbeit aufbauen oder Teile dieser Arbeit einbinden und der Betreuer dieser Arbeit. Die Betreiber öffentlicher Parks haben ein Interesse daran, dass die Kollisionsvermeidung funktioniert und ein problemloser Fortbewegungsablauf in den Parks stattfindet.

Ersteller der Arbeit: Der Ersteller dieser Arbeit hat ein großes Interesse daran fachliche Erkenntnisse zu erlangen und dass die Kollisionsvermeidung funktioniert. Zusätzlich liegt ein Interesse darin, dass das System wartbar ist, auf andere Roboter übertragen werden kann und in weiterführenden Arbeiten oder anderen Projekten eingesetzt werden kann.

Weiterführende Forschungsarbeiten: Forschungsarbeiten, die auf dieser Arbeit aufbauen oder Teile dieser Arbeit einbinden, haben ebenfalls ein Interesse an der Funktionalität und Portierbarkeit des Systems. Sie haben zusätzlich ein Interesse daran, nicht zwingend das gesamte System zu übernehmen, sondern nur bestimmte Teile oder Funktionalitäten. Eine weiterführende Forschungsarbeit kann das in Kapitel 1 eingeführte Projekt zur autonomen Müllbeseitigung, begleitet durch Prof. Dr. Dahlkemper, sein.

Prüfer: Der betreuende Erstprüfer Prof. Dr. Hensel hat neben der Funktionalität und Erweiterbarkeit ein Interesse an der Wiederverwendbarkeit der eingesetzten Hardware, dem Zugewinn des erlangten Wissens und der Einhaltung des vorgegebenen Budgets.

Um die Interessen der Stakeholder aus Tabelle 4.1 zu berücksichtigen, werden die Interessengebiete in konkrete Anforderungen überführt, welche in Abschnitt 4.5 formuliert werden. Sollten sich Anforderungen gegenseitig ausschließen, wird abhängig von der Wichtigkeit entschieden, welche Anforderung berücksichtigt wird.

Tabelle 4.1: Analyse der Stakeholder

ID	Stakeholder	Interessengebiet
1	Betreiber des Parks	Anforderungen an die Funktionalität des Gesamtsystems
2	Personen im Park	Anforderungen an die Funktionalität der Kollisionsvermeidung und an personenbezogene Daten
3	Ersteller der Arbeit	Anforderungen an die Funktionalität und die die Einsetzbarkeit des Systems
4	Weiterführende Forschungsarbeiten	Anforderungen an die Funktionalität, die Erweiterbarkeit, die Portierbarkeit des Systems und gesammelte Daten
5	Betreuender Erstprüfer	Anforderungen an die Funktionalität, die Wiederverwendbarkeit verwendeter Materialien, erlangter Kenntnisse und Quellcode, den Zugewinn von erlangtem Wissen und an die Kosten
6	Bediener des Systems	Anforderung an die Zuverlässigkeit und Bedienbarkeit des Systems
7	DSGVO	Anforderungen an die Verarbeitung personenbezogener Daten

4.3 Anwendungsfälle

Nach der Abgrenzung des Systems werden nun die Anwendungsfälle für die Kollisionsvermeidung analysiert. In Abbildung 4.2 ist ein UML-Anwendungsfall-Diagramm dargestellt, in welchem die verschiedenen Anwendungsfälle abgebildet sind. Die Anwender beziehen sich dabei auf die Komponenten des Systemkontextes aus Abschnitt 4.1. Die wichtigsten Anwender für die Funktionalität des Systems sind die Objekte, die von dem System zu erkennen sind und dessen Bewegungen berechnet werden müssen. Die DSGVO und die Lichtverhältnisse stellen Anforderungen an die Sensoren und die Datenverarbeitung, die bei der Entwicklung zu berücksichtigen sind. Für benachbarte Systeme, wie die Robotersteuerung, muss eine Schnittstelle definiert werden, um eine Kollisionsmeldung zu übertragen. Im Folgenden werden die Anwendungsfälle separat beschrieben.

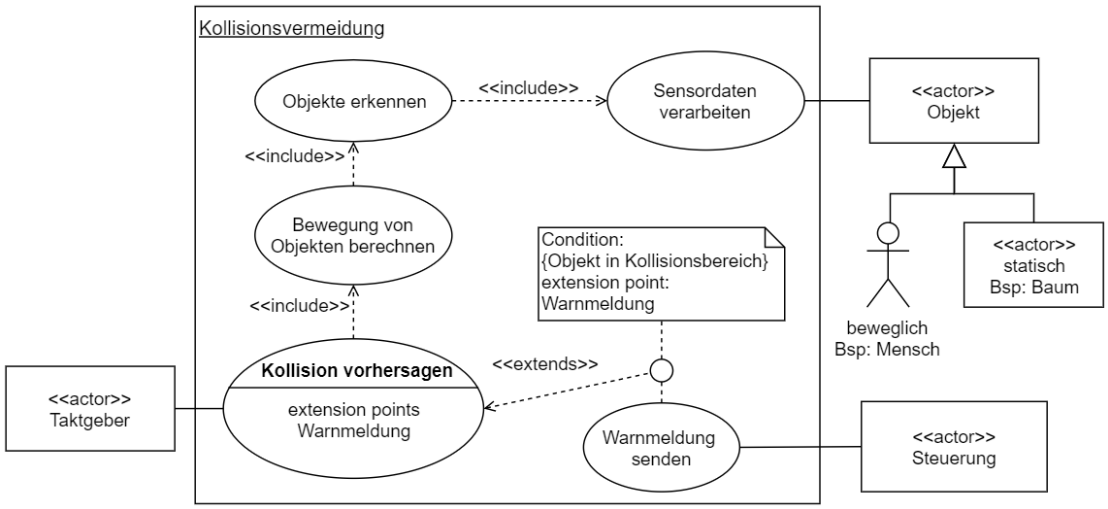


Abbildung 4.2: Anwendungsfall-Diagramm der Kollisionsvermeidung

Kollision vorhersagen

Über einen vorgegebenen festen Takt, wird die Wahrscheinlichkeit einer Kollision des Roboters mit erkannten Objekten berechnet. Für die Einstufung der Wahrscheinlichkeit einer Kollision werden vorher die Bewegungseigenschaften der Objekte berechnet. Wird die Wahrscheinlichkeit hoch genug eingestuft, wird eine Gefahrenmeldung an die Steuerung gesendet.

Tabelle 4.2: Kollision vorhersagen

Name	Kollision vorhersagen.
Beschreibung	Von den erkannten Objekten werden die Bewegungen berechnet.
Ergebnis	Warnmeldung über eine bevorstehende Kollision.
Vorbedingung	Es ist eine Verbindung mit der Steuerung aufgebaut.
Auslösendes Ereignis	Vorgegebener zeitlicher Takt.
Standardablauf	1. Einlesen der Objektinformationen. 2. Berechnung der Wahrscheinlichkeit einer Kollision. 3. Senden einer Warnmeldung, wenn sich ein Objekt im Kollisionsbereich befindet.

Bewegung erkannter Objekte berechnen

Für jedes erkannte Objekt wird die Bewegungsrichtung und die Bewegungsgeschwindigkeit berechnet. Die berechneten Informationen werden anschließend für die Vorhersage einer Kollision verwendet. Die Berechnung der Informationen wird dann gestartet, wenn Objekte erkannt wurden.

Tabelle 4.3: Bewegung erkannter Objekte berechnen

Name	Bewegung erkannter Objekte berechnen.
Beschreibung	Von den erkannten Objekten werden die Bewegungen berechnet.
Ergebnis	Liste mit Objekten und die zugehörigen Bewegungszustände.
Vorbedingung	Es wurden Objekte erkannt.
Nachbedingung	Warten auf neue Objekte.
Auslösendes Ereignis	Der Anwendungsfall Kollision vorhersagen fordert neue Bewegungszustände an.
Standardablauf	1. Einlesen der erkannten Objekte. 2. Die Bewegungszustände der Objekte werden berechnet.

Objekte erkennen

Das Erkennen von Objekten erfolgt nach dem Verarbeiten der Sensordaten. Die Objekterkennung wird ausgeführt und erzeugt eine Liste mit erkannten Objekten.

Tabelle 4.4: Objekte erkennen

Name	Objekte erkennen.
Beschreibung	Die verarbeiteten Sensordaten werden verwendet, um Objekte in der Umgebung des Roboters zu detektieren.
Ergebnis	Liste mit erkannten Objekten.
Vorbedingung	Die Verarbeitung der Sensordaten ist abgeschlossen.
Auslösendes Ereignis	Der Anwendungsfall Bewegung von Objekten berechnen fordert eine Liste mit erkannten Objekten.
Standardablauf	1. einlesen der verarbeiteten Sensordaten 2. Ausführen des Objekterkennungsalgorithmus

Sensordaten verarbeiten

Die Verarbeitung von Sensordaten bereitet die rohen Daten des verwendeten Sensors auf. Dies kann erfolgen, wenn eine Verbindung zu dem Sensor aufgebaut ist.

Tabelle 4.5: Anwendungsfall: Sensordaten verarbeiten

Name	Sensordaten verarbeiten.
Beschreibung	Die Daten von den Sensoren werden eingelesen und so aufbereitet, dass sie verwendet werden können, um Objekte zu erkennen.
Ergebnis	Aufbereitete Sensordaten
Vorbedingung	Die Kommunikation mit dem Sensor ist aktiv.
Auslösendes Ereignis	Der Anwendungsfall Objekte erkennen fordert neue Sensordaten an.
Standardablauf	1. Die Sensordaten werden eingelesen. Die Sensordaten werden verarbeitet.

4.4 Rechtliche Einschränkungen

Bei dem Einsatz von mobilen Robotern müssen verschiedene rechtliche Regelungen eingehalten werden. Kommen in einem Roboter Sensoren zum Einsatz, die personenbezogene Daten verarbeiten, muss die Datenverarbeitung innerhalb der DSGVO stattfinden [14]. In einer von der TU Berlin in Auftrag gegebenen rechtlichen Studie [14] wurde unter anderem die Verarbeitung personenbezogener Daten von Robotern analysiert. Nach der

DSGVO fällt nahezu jeder Umgang mit personenbezogenen Daten unter das Datenschutzrecht. Dazu gehört auch das Einlesen und anschließend direkte Löschen von Sensordaten, die personenbezogene Daten enthalten. In der Studie [14] werden verschiedene Szenarien genannt, in denen personenbezogene Daten verarbeitet werden. Die für diese Arbeit relevanten Szenarien sind in der folgenden Auflistung aufgeführt:

- Der Roboter wird im Rahmen eines Forschungsvorhabens erprobt, wodurch angenommen werden kann, dass Personen, die an dieser Erprobung teilnehmen, eine personenbezogene Datenerfassung erwarten. Dies gilt nicht nur für informierte Probanden, sondern dürfte allgemein auf Personen im Erprobungskontext erweitert werden.
- Es muss ein berechtigtes Interesse zur Verarbeitung der Daten auf Seiten des Verantwortlichen bestehen. Dies gilt auch für den störungsfreien und sicheren Betrieb des Roboters. Dabei muss jedoch sichergestellt sein, dass die Verwendung eines Sensors zur Erfassung personenbezogener Daten notwendig ist. Als Beispiel wird hier die Kollisionsvermeidung angeführt, die nicht zwingend mit einer Kamera durchgeführt werden muss, sondern auch mit Lidar- oder Radarsensoren durchgeführt werden kann.
- Das System wird so konzipiert, dass der Informationsgehalt der Daten so weit minimiert wird, dass mit ihm keine Personen mehr identifizierbar sind. Dieser Punkt zeigt sich jedoch als schwierig, da für das unkenntlich machen, die Daten trotzdem erhoben werden müssen.

4.5 Formulierung der Anforderungen

In diesem Abschnitt werden die Anforderungen an die Kollisionsvermeidung gestellt. Die Anforderungen werden unterteilt in Funktionale Anforderungen, welche sich aus den Anwendungsfällen ergeben, in Anforderungen an das System, welche die Funktionalen Anforderungen erweitern und in weitere Anforderungen, die sich aus der Stakeholderanalyse und den rechtlichen Einschränkungen ergeben. Die Anforderungen werden in die Priorität *muss* und *soll* eingeteilt. Anforderungen mit der Priorität *muss* müssen für das gelingen der Arbeit erfüllt sein. Anforderungen mit der Priorität *soll* stellen zusätzliche Anforderungen, die zu berücksichtigen sind, jedoch nicht von dem Gelingen der Arbeit abhängen.

4.5.1 Funktionale Anforderungen

Die Anforderungen, die sich aus den Anwendungsfällen ableiten lassen, werden in diesem Abschnitt vorgestellt und in Tabelle 4.6 zusammengefasst. Die Anforderungen an die Funktionalität müssen erfüllt werden, um erfolgreich eine Kollisionsvermeidung durchzuführen. Aus diesem Grund werden alle Anforderungen mit der Priorität *muss* eingestuft. Um die Kollisionsvorhersage ausführen zu können, müssen Objekte, die eine Kollision hervorrufen können, erkannt werden. Es muss also eine Objekterkennung umgesetzt werden. Um die Kollision für einen Prädiktionshorizont *dt* vorherzusagen, ist es notwendig die Bewegungseigenschaften der Objekte zu berechnen. Dafür muss eine Objektverfolgung umgesetzt werden, die die Bewegungseigenschaften aus den vergangenen Objekten berechnet. Aus den Bewegungseigenschaften und *dt* muss die Wahrscheinlichkeit einer Kollision vorhergesagt werden können und die Informationen an die Steuerung gesendet werden. Dafür muss eine Schnittstelle geschaffen werden, über die Informationen an die Steuerung gesendet werden.

Tabelle 4.6: Funktionale Anforderungen

ID	Priorität	Anforderung
1	muss	Das System muss Objekte erkennen, die eine Kollision verursachen können.
2	muss	Das System muss vorhersagen können, ob eine Kollision bevorsteht oder nicht.
3	muss	Das System muss Objekte verfolgen können.
4	muss	Das System muss Informationen an die Steuerung senden können.

4.5.2 Anforderungen an das System

In diesem Abschnitt werden weitere Anforderungen formuliert, die nicht allein aus den Anwendungsfällen hergeleitet werden können. Die Anforderungen erweitern die Anforderungen an die Funktionalität, sie beinhalten Anforderungen an die Hard- und Software. Die Anforderungen sind in Tabelle 4.7 festgehalten.
Aus dem Systemkontxt geht hervor, dass starre und bewegliche Hindernisse erkannt werden müssen. Um eine Kollision zu vermeiden, muss das System rechtzeitig eine Warnung generieren. Es wird davon ausgegangen, dass Fahrradfahrer die schnellsten Objekte sind und sich aufgrund der Umgebung nicht schneller als 15 km/h bewegen, was $4,16$ m/s entspricht. Der Roboter bewegt sich mit $0,2$m/s, was eine Gesamtgeschwindigkeit von

$v_{max} = 4,36\text{m/s}$ ergibt. Um der Steuerung die Möglichkeit zu geben, Objekten auszuweichen, müssen Kollisionen möglichst frühzeitig erkannt werden. Da sich der Roboter langsam bewegt, wird der zeitliche Horizont für die Reaktion dt auf 2s festgelegt. Das gibt der Steuerung genug Zeit, langsamen und starren Objekten auszuweichen. Um eine Kollision für Objekte, die sich mit der maximalen Geschwindigkeit bewegen, für den gesamten Horizont festlegen zu können, muss der Sensor mindestens eine Reichweite von

$$z_{max} = 2s \cdot 4,36m/s = 8,72m \tag{4.1}$$

aufweisen. Neben schnellen Objekten, die sich weit weg befinden, müssen auch Objekte erkannt werden, die sich nah vor dem Roboter befinden. Es wird davon ausgegangen, dass die Objekte nicht unmittelbar vor dem Roboter auftauchen, sondern sich in das Das FOV bewegen. Ausgehen von der Größe des Roboters und der Umgebung, wird davon ausgegangen, dass sich Objekte mit einem Abstand von mindestens $0,4\text{m}$ in das Sichtfeld des Sensors bewegen.
Neben der Entfernung wird die Anforderung an den Öffnungswinkel des Sensors gestellt. Es müssen Objekte erkannt werden, die entlang der Fahrtrichtung des Roboters auftreten können. Da sich die Umgebung auf öffentliche Parks bezieht, ist es möglich, dass sich Objekte seitlich der Fahrtrichtung nähern. Der Öffnungswinkel muss so groß gewählt werden, dass der gesamte Bereich entlang der Fahrtrichtung inklusive einer Gefahrenzone abdeckt.
Die Abtastrate soll so hoch gewählt werden, dass das System Objekte, die sich mit der angenommenen Maximalgeschwindigkeit bewegen verfolgen kann. Diese Anforderung ist mit der Priorität soll eingestuft, da der Referenzroboter nicht in der Lage ist, einem Objekt auszuweichen, dass sich mit z_{max} auf ihn zu bewegt.

4.5.3 Anforderungen der Stakeholder

In diesem Abschnitt werden Anforderungen aufgeführt, die sich durch die Stakeholderanalyse ergeben. Die Anforderungen sind in Tabelle 4.8 aufgelistet. Die bereits durch den Systemkontext oder die Anwendungsfälle hergeleiteten Anforderungen, die sich auch aus den Stakeholdern ergeben, werden nicht erneut aufgeführt.
Für die Personen, die sich im Park befinden, und die DSGVO ergibt sich die Anforderung an die Einhaltung des Datenschutzes. Es wird die Anforderung an das System gestellt, keine Personenbezogenen Daten zu verarbeiten.

Tabelle 4.7: Anforderungen an das System

ID	Priorität	Anforderung
5	soll	Das System soll Objekte mit einer Geschwindigkeit von z_{max} verfolgen können.
6	soll	Das System soll mehrere Objekte gleichzeitig erkennen und verfolgen.
7	muss	Der Sensor muss für den Outdooreinsatz geeignet sein.
8	muss	Der Sensor muss einen Arbeitsbereich von 0,4m bis 8,72m vor dem Roboter aufweisen.
9	muss	Das System muss Objekte in dem Gefahrenbereich entlang der Fahrtrichtung des Systems erkennen.
10	muss	Das System muss Objekte in dem gesamten Arbeitsbereich erkennen können.

Um das Interesse der weiterführenden Forschungsarbeiten zu berücksichtigen, wird die Anforderung gestellt, dass die einzelnen Module, Schnittstellen aufweisen müssen, um sie in anderen Projekten einsetzen zu können. Zudem sollen die Module portierbar sein, also auf gängigen Betriebssystemen einsetzbar sein. Als gängige Betriebssysteme werden Linux, Apple OSX und Windows festgelegt.
Der betreuende Erstprüfer stellt die Anforderung, dass das Budget von 500€ eingehalten wird, die erlangten Kenntnisse dokumentiert werden und der Quellcode verfügbar ist. Zudem muss die verwendete Hardware für nachfolgende Projekte wiederverwendbar sein.

Tabelle 4.8: Zusätzliche Anforderungen der Stakeholder

ID	Priorität	Anforderung
11	muss	Das System muss sich innerhalb der DSGVO nach [14] bewegen.
12	soll	Die Software soll in separat ausführbare Module unterteilt werden.
13	soll	Die Module sollen über feste Schnittstellen ansprechbar sein.
14	soll	Die Software soll auf den Betriebssystemen Linux, Windows und OSX ausführbar sein.
15	muss	Neuanschaffungen dürfen das Budget von 500€ nicht überschreiten
16	muss	Die erlangten Erkenntnisse müssen dokumentiert werden.
17	soll	Die verwendete Hardware muss wiederverwendbar sein.

5 Konzeption

In diesem Kapitel wird das Konzept für die Umsetzung vorgestellt, welches die in Kapitel 4 gestellten Anforderungen berücksichtigt. Zuerst wird ein Konzept für die Hardware vorgestellt. Dazu wird aus verschiedenen Sensoren ein geeigneter ausgewählt, welcher den Referenzroboter ExoMy erweitert. Um die Sensoren miteinander vergleichen zu können, wird für jede Eigenschaft eine Bewertung mit „ + “ für gut geeignet, „ o “ mittelmäßig geeignet und „ - “ schlecht geeignet vorgenommen. Anhand dieser Bewertung wird anschließend ein Sensor ausgewählt. Anhand des ausgewählten Sensors und seiner Eigenschaften wird anschließend ein Konzept für die Umsetzung der Software vorgestellt.

5.1 Hardware

Die Auswahl des Sensors erfolgt in zwei Schritten, zuerst wird der Sensorstyp und anschließend der Sensor selbst festgelegt.

5.1.1 Vergleich verschiedener Sensortypen

In Abschnitt 2.5 wird der Roboter ExoMy vorgestellt. Dieser Roboter soll um ein Kollisionsvermeidungssystem erweitert werden und dabei möglichst wenig Änderungen an der Hardware vorgenommen werden. Für die Hardware wird der Ansatz verfolgt, den Roboter um einen Sensor zu erweitern, der die nötigen Informationen für eine Kollisionsvorhersage bereitstellt. Zur Auswahl stehen Lidar-Sensoren, Radarsensoren und Tiefenkameras. In Tabelle 5.1 sind die Eigenschaften der Sensoren dargestellt und werden anhand der Anforderungen bewertet.
Das FOV wird aufgeteilt in die Entfernung und den Winkel des Sichtfeldes. Sowohl Standard-Lidar- als auch Radarsensoren haben ein schmales Sichtfeld, können dafür aber sehr gut große Entfernungen erfassen. Der Arbeitsbereich von Radarsensoren ist jedoch in der Regel nicht geeignet, um Objekte in einer kurzen Entfernung wahrzunehmen.

Tabelle 5.1: Vergleich verschiedener Sensoren

Kriterium	Lidar	Radar	Tiefenkameras
FOV Entfernung	+	+	o
FOV Winkel	-	-	+
Abtastfrequenz	+	+	+
Genauigkeit	+	-	+
Lichtempfindlichkeit	o	+	o
Wiederverwendbarkeit	o	o	+
Veränderung des Roboters	+	+	+
Auswahl Sensor			x

Kameras hingegen haben in der Regel einen größen Öffnungswinkel, können aber weite Entfernungen nicht gut wahrnehmen. Die Frequenz ist von allen Sensoren ausreichend groß, wobei mit dem Radar und Lidar deutlich höhere Abtastraten möglich sind als mit herkömmlichen Kameras. Bei der Genauigkeit und der Robustheit gegen schlechte Sicht verhalten sich der Lidar und die Kamera relativ ähnlich, da beide Sensoren mit einfallendem Licht arbeiten, haben beide bei beispielsweise Nebel einen deutlichen Nachteil gegenüber dem Radarsensor. Lidar und Kameras sind jedoch deutlich genauer als der Radarsensor. Die Wiederverwendbarkeit von Kameras ist Aufgrund der vielseitigen Einsetzbarkeit deutlich besser einzustufen. Die Veränderung des Roboters ist bei allen Sensoren mit dem gleichen sehr geringen Aufwand verbunden.
Das entscheidende Kriterium für die Auswahl des Sensors ist das FOV. In Parks ist es möglich, dass sich Objekte aus verschiedenen Richtungen auf den Roboter zu bewegen. Für den Fall, dass sich Objekte von der Seite nähern ist ein großer Öffnungswinkel notwendig. Da Tiefenkameras den größten Öffnungswinkel aufweisen, wird in dieser Arbeit eine Tiefenkamera eingesetzt.

5.1.2 Vergleich von Tiefenkameras

Im nächsten Schritt wird genauer spezifiziert, welche Tiefenkamera eingesetzt Einsatz wird. Es gibt verschiedene Anbieter von Tiefenkameras. Vor allem im industriellen Bereich gibt es mehrere Anbieter. Da diese häufig sehr teuer sind oder spezielle Software und Hardware benötigen, wird eine Kamera gesucht, die möglichst gängige Soft- und Hardwareschnittstellen verwendet. Die Firma Intel bietet mit der Realsense Produktfamilie eine Reihe von Tiefenkameras, die genau diese Aspekte berücksichtigen. Durch das Realsense SDK 2.0 ist die Verwendung mit verschiedenen Betriebssystemen wie Windows, Linux,

Android und macOS möglich. Zudem ist es möglich Anwendungen mit verschiedenen Programmiersprachen wie Python oder C++ zu erstellen. Zusätzlich wird die Realsense SDK 2.0 in verschiedenen Frameworks wie ROS, ROS2, Open3D oder OpenCV eingesetzt. In Tabelle 5.2 sind die verschiedenen Tiefenkameras und ihre für diese Arbeit relevanten Eigenschaften dargestellt. Die Informationen zu den Kameras stammen zum einen aus dem Datenblatt [17] und zum anderen von der Realsense Website [16]. In der Tabelle sind nicht alle Realsense Modelle aufgeführt, da nur Kameras relevant sind, die sich für den Außeneinsatz eignen. Alle aufgeführten Kameras verfügen über einen Tiefensensor und über einen RGB-Sensor. Die Tiefensensoren der D455 und D435 Kamera haben ein breites FOV, wodurch sie viel von der Umgebung in Fahrtrichtung aufnehmen können. Das FOV des RGB-Sensors von der D435 ist jedoch kleiner, als das FOV des Tiefensensors, was bei einer Fusion der Bildinformationen zu einer Verkleinerung des Gesamt-FOV führt. Wird der RGB-Sensor beispielsweise zur Validierung verwendet, kann bei der D435 nur ein Ausschnitt des Tiefenbildes ausgewertet werden. Die D415 hat ein geringeres FOV als die anderen Kameras, jedoch die gleiche Pixelanzahl. Dadurch ist die Auflösung des aufgenommenen Bereiches höher. Der gesamte Arbeitsbereich ist für alle Kameras mit bis zu 10m angegeben. Die Genauigkeit der Kameras nimmt mit größer werdenden Entfernungen ab. Für die D455 ist ein optimaler Arbeitsbereich bis 6m und ein Tiefenfehler von weniger als 2% für eine Entfernung von bis zu 4m angegeben. Für die anderen Kameras liegt der optimale Einsatzbereich nur bei bis zu 3m Entfernung und der Tiefenfehler nur bei bis zu 2m unter 2%. Aufgrund des größeren FOV und der höheren Genauigkeit bei größeren Entfernungen wird in dieser Arbeit die Intel Realsense D455 Tiefenkamera verwendet.

Tabelle 5.2: Vergleich von Intel Tiefenkameras

Kriterium	D455	D435i/D435	D415
Tiefentechnologie	Stereoskopie	Stereoskopie	Stereoskopie
FOV RGB	90° x 65°	69° x 42°	69° x 42°
Auflösung RGB	1280x800	1920x1080	1920x1080
FOV Tiefe	87° x 58°	87° x 58°	65° x 40°
Auflösung RGB	1280x720	1280x720	1280x720
Idealer Arbeitsbereich	0,6 m - 6 m	0,3 m - 3 m	0,5 m - 3 m
gesamter Arbeitsbereich	0,4 m - 10 m	0,2 m - 10 m	0,3 m - 10 m
Tiefenfehler	<2% bis 4 m	<2% bis 2 m	<2% bis 2 m

5.2 Software

Nach der Auswahl der Hardwarekomponenten wird in diesem Abschnitt das Konzept für die Software vorgestellt. Zuerst wird die Entwicklungsumgebung festgelegt und anschließend der Ablauf der Kollisionsvermeidung skizziert. Aus dem Ablauf werden Funktionalitäten abgeleitet und anschließend Konzepte für die einzelnen Funktionen vorgestellt. Die Konzepte für die einzelnen Schritte werden nicht nach der Abarbeitungsreihenfolge vorgestellt, sondern in der Reihenfolge, wie sie konzipiert werden.

5.2.1 Entwicklungsumgebung

In diesem Teilabschnitt wird erläutert, in welcher Umgebung die Software entwickelt wird und welche unterstützenden Bibliotheken und Frameworks zum Einsatz kommen.
Um die Anforderungen an die Software in Tabelle 4.8 zu erfüllen, wird bei der Entwicklung der Software ROS2 verwendet. Das bietet die Möglichkeit die Software in Packages zu unterteilen und diese im späteren Verlauf zu erweitern oder in anderen mit ROS2 entwickelten Projekten zu integrieren.
Von dem Kamerahersteller Intel werden Bibliotheken für Python und C++ bereitgestellt, mit denen die Kamera direkt angesprochen, Bilder ausgelesen und Einstellungen vorgenommen werden können. Als Unterstützung bei der Bildverarbeitung bietet es sich an, die frei zugängliche Bibliothek OpenCV zu verwenden, welche effiziente Algorithmen zur Bildverarbeitung bereitstellt. Sowohl OpenCV als auch ROS2 ermöglichen die Entwicklung mit C++ und Python.
C++ hat den Vorteil, dass die Ausführungszeiten der Programme in der Regel schneller sind, als in Python. Um diesen Nachteil auszugleichen, gibt es in Python die Möglichkeit, Bibliotheken wie NumPy oder auch OpenCV zu verwenden, welche Funktionen enthalten, in denen der Quellcode oder ein Teil davon bereits in C-Code kompiliert ist. Ein Vorteil von Python ist die sehr große Community, welche auf verschiedenen Plattformen Antworten zu verschiedenen Fragestellungen bereithält. Für die Umsetzung dieser Arbeit sind folglich sowohl Python als auch C++ geeignet. Aufgrund der größeren Community, der stärkeren Verbreitung und des Interesses des Entwicklers wird in dieser Arbeit Python verwendet.

5.2.2 Ablauf der Kollisionsvermeidung

Im Folgenden wird der grobe Ablauf der Kollisionsvermeidung konzipiert. Der Ablauf ist in Abbildung 5.1 dargestellt. Zu Beginn muss die Kollisionsvermeidung initialisiert

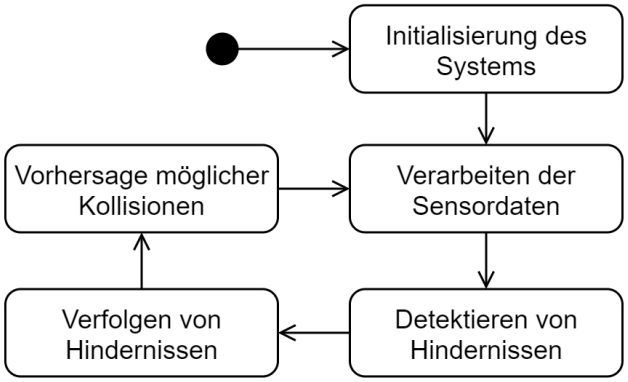


Abbildung 5.1: Aktivitätsdiagramm der Kollisionsvermeidung

werden. Anschließend werden die 3D-Bilddaten der Stereokamera ausgelesen und verarbeitet (Sensordatenverarbeitung). Um die Anforderung an die DSGVO nicht zu verletzen, werden die RGB-Daten nicht verwendet. Aus den verarbeiteten 3D-Bilddaten werden Objekte, welche für den Roboter als Hindernisse zu kennzeichnen sind, detektiert (Objekterkennung). Um die Gefahr einer Kollision mit den erkannten Objekten möglichst genau bestimmen zu können, werden die Bewegungszustände der Objekte berechnet (Objektverfolgung). Mit den errechneten Bewegungszuständen wird im letzten Schritt die Wahrscheinlichkeit einer Kollision geschätzt und die Einstufung an die Steuerung gesendet (Kollisionsvorhersage). Anschließend startet der Ablauf bei dem Einlesen der Sensordaten neu.
Werden von der Objekterkennung keine Objekte gefunden, werden die Schritte Objektverfolgung und Kollisionsvorhersage trotzdem ausgeführt. Werden keine Objekte gefunden, werden die Objekte des vorherigen Schrittes aus der Objektverfolgung gelöscht und die Kollisionsvorhersage gibt die Meldung keine Gefahr aus. Werden Objekte zum ersten Mal erkannt, werden die Objekte in der Objektverfolgung mit der Geschwindigkeit null initialisiert und im nachfolgenden Durchlauf die Geschwindigkeit berechnet.
Aus dem Ablauf lassen sich die vier bereits hervorgehobenen Arbeitsschritte Sensordatenverarbeitung, Objekterkennung, Objektverfolgung und Kollisionsvorhersage ableiten. Diese Arbeitsschritte stellen in dieser Arbeit die vier Hauptkomponenten der Kollisionsvermeidung dar.

5.2.3 Kollisionsvorhersage

In [27] wird ein Ansatz zur Kollisionsvorhersage vorgestellt, bei dem eine Region of Interest (ROI) festgelegt wird, welche als *gefährlicher Bereich* definiert ist. Befindet sich ein Objekt in diesem Bereich wird die Meldung *Gefahr* ausgegeben. Bewegt sich ein Objekt in die Richtung der ROI und erreicht sie innerhalb eines vorgegebenen zeitlichen Horizonts dt, wird die Meldung *Potentielle Gefahr* ausgegeben. Andernfalls wird die Meldung *Keine Gefahr* ausgegeben. Die Größe der ROI orientiert sich in [27] an der Geschwindigkeit und der Fahrtrichtung des Fahrzeugs.
Der Ansatz wird in dieser Arbeit aufgegriffen und angepasst. Es wird ebenfalls eine ROI festgelegt, welche als *gefährlicher Bereich* definiert ist. Die Einstufung der Gefahr einer Kollision erfolgt dabei nach dem gleichen in [27] vorgestellten Prinzip. Da sich der Roboter sehr langsam bewegt, wird die ROI statisch an die Abmessungen des Roboters gekoppelt. Die Abmessungen der ROI sind für die Achsen definiert durch

$$ROI_x = ROI_{x\_min} \rightarrow ROI_{x\_max} \tag{5.1}$$

$$ROI_y = -\infty \rightarrow ROI_{y\_max} \tag{5.2}$$

$$ROI_z = ROI_{z\_min} \rightarrow ROI_{z\_max}. \tag{5.3}$$

Da sich der Roboter auf dem Boden bewegt, was dem Koordinatenursprung der y-Achse entspricht, und sich Objekte nicht unterhalb des Bodens bewegen können, wird für die y-Achse nur eine obere Grenze ROI_y festgelegt. ROI_y wird über die Höhe des Roboters und einen Sicherheitsabstand definiert. Für die x, z-Ebene werden jeweils zwei Grenzen für jede Achse gewählt. Diese Grenzen werden durch die Abmessungen des Roboters und einem zusätzlichen Sicherheitsabstand festgelegt. Der Roboter mit der zugehörigen ROI ist in Abbildung 5.2 für die x, z-Ebene $ROI_{x,z}$ dargestellt. $ROI_{x,z}$ wird von acht weiteren Bereichen umgeben, welche in Fahrtrichtung des Roboters mit den Himmelsrichtungen bezeichnet sind. $ROI_{x,z}$ lässt sich vollständig mit den vier Eckpunkten *(NO, NW, SW, SO)* aus Abbildung 5.2, rot gekennzeichnet, beschreiben. Die Eckpunkte setzen sich jeweils aus der z- und x-Koordinate der festgelegten Grenzen durch

$$NO = (ROI_{z\_max}, ROI_{x\_min}) \tag{5.4}$$

$$NW = (ROI_{z\_max}, ROI_{x\_max}) \tag{5.5}$$

$$SO = (ROI_{z\_min}, ROI_{x\_min}) \tag{5.6}$$

$$SW = (ROI_{z\_min}, ROI_{x\_max}) \tag{5.7}$$

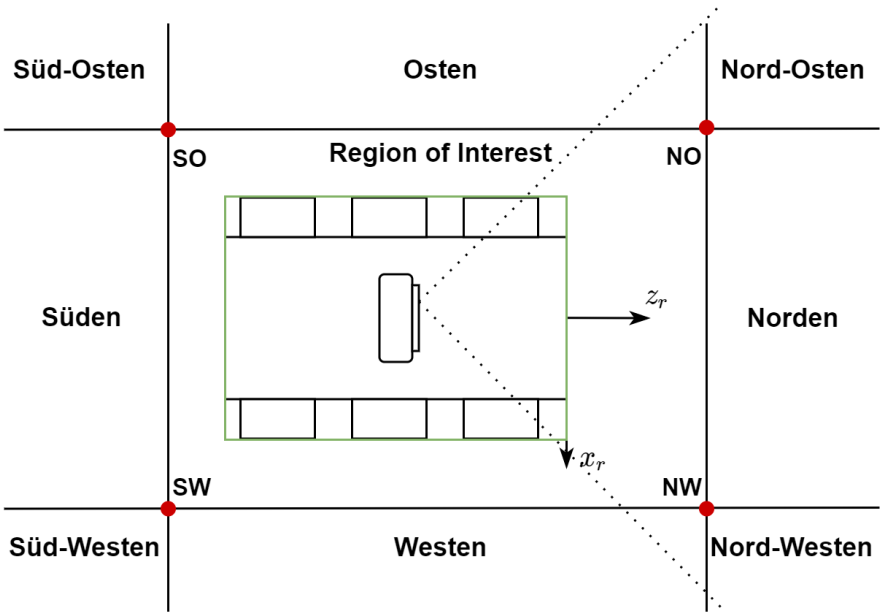


Abbildung 5.2: Region of Interest des Roboters

zusammen. Neben den Bereichen ist in der Abbildung das FOV der Kamera dargestellt. Da die Kamera in Fahrtrichtung ausgerichtet ist, einen kleineren Öffnungswinkel als 90° aufweist und fest mit dem Roboter verbunden ist, kann im Folgenden davon ausgegangen werden, dass keine Objekte erkannt werden, die sich im mit *Süden* gekennzeichneten Bereich befinden.
Um eine Aussage über eine Kollision treffen zu können, benötigt die Kollisionsvorhersage Informationen über die aktuelle Position, die Positionsänderung und die Größe der verfolgten Objekte, welche im Folgenden eingeführt werden.

Beschreibung der Objekte Der in Teilabschnitt 3.3.2 vorgestellte SORT-Algorithmus [2] verwendet die Bounding Box, um die Objekte für die Objektverfolgung zu beschreiben. Diese Beschreibung wird an dieser Stelle aufgegriffen und für den dreidimensionalen Raum erweitert.
Da sich die Kollision auf die hardwareseitige Systemgrenze, den Roboter bezieht, werden die Informationen im Roboterkoordinatensystem angegeben. Für die Beschreibung der Position werden die Koordinaten (u', v') durch (x_r, y_r, z_r) ersetzt, welche die Position der linken unteren Ecke beschreiben. Für die Geschwindigkeit wird über die Änderung der Position $(\dot{u}, \dot{v})$ durch $(\dot{x}_r, \dot{y}_r, \dot{z}_r)$ ersetzt. Für die Beschreibung der Größe der Objektes werden die Parameter (w, h) durch (w_r, h_r) ersetzt. Der resultierende Vektor zum

Beschreiben der Objekte $\mathbf{x}_{predict}$ für die Kollisionsvorhersage ergibt sich zu

$$\mathbf{x}_{predict} = \begin{pmatrix} x_r & y_r & z_r & \dot{x}_r & \dot{y}_r & \dot{z}_r & w_r & h_r \end{pmatrix}^T. \tag{5.8}$$

Überprüfung auf *Gefahr* Im Folgenden wird beschrieben, wie überprüft wird, ob sich ein Objekt in der ROI befindet. Für die Überprüfung werden die Eckpunkte der ROI und der Zustandsvektor $\mathbf{x}_{predict}$ verwendet. Befindet sich ein Objekt nördlich von ROI_z, östlich oder westlich von ROI_x oder höher als ROI_y, so ist das Objekt außerhalb der ROI. Andernfalls befindet sich das Objekt für jede Koordinate innerhalb der ROI und die Einstufung *Gefahr* wird vorgenommen.

Überprüfung auf *Potentielle Gefahr* In diesem Abschnitt wird überprüft, ob sich ein Objekt innerhalb eines vorgegebenen Horizonts dt in die ROI bewegt oder nicht. Es ist möglich, dass sich ein Objekt zu Beginn und zum Ende des Horizonts außerhalb der ROI befindet, aber zwischenzeitlich innerhalb. Aus diesem Grund ist es notwendig zu überprüfen, ob sich das Objekt zu einem beliebigen Zeitpunkt von dt innerhalb der ROI befindet.
Aus Abschnitt 4.1 geht hervor, dass sich der Roboter in öffentlichen Parks bewegt. Ausgehend davon wird im Folgenden angenommen, dass es sich die Objekte auf dem Boden befinden. Für Objekte, die sich oberhalb oder im Grenzbereich von ROI_y befinden, wird angenommen, dass es sich um statische Objekte wie Äste oder Laternen handelt. Aus diesem Grund wird für die y-Achse nur überprüft, ob sich y_r zum Startpunkt oder zum Endpunkt des Horizonts innerhalb der ROI befindet. Befindet sich y_r zu beiden Zeitpunkten außerhalb der ROI, wird die Meldung *keine Gefahr* generiert. Für die Überprüfung muss die Position für das Ende des Horizonts y_{dtr} berechnet werden. Mit der Annahme, dass die Geschwindigkeit der Objekte konstant ist, berechnet sich y_{dtr} mit

$$y_{dtr} = y_r + \dot{y_r} \cdot dt. \tag{5.9}$$

Ansonsten wird für die x, z-Ebene überprüft, ob sich das Objekt innerhalb von dt in die ROI bewegt.
Es wird davon ausgegangen, dass sich die Objekte in der x, z-Ebene deutlich schneller bewegen als entlang der y-Achse. Aus diesem Grund wird für diese Ebene für jeden Zeitpunkt des Horizonts überprüft, ob sich ein Objekt in der ROI befindet. Für die x, z-Ebene wird der Schnittpunkt der Bounding Box mit der ROI anhand des Schnittpunktes zweier

Geraden nach [5] bestimmt.
Schneiden sich zwei der Geraden im Bereich der ROI und innerhalb von dt, wird die Einstufung *Potentielle Gefahr* vorgenommen. Für die Überprüfung der Schnittpunkte werden jeweils die Geraden G, bestehend aus den Eckpunkten der ROI, und den Geraden G', bestehend aus den Eckpunkten der Bounding Box des Objektes, gebildet. Für $G(P_1, P_2)$ werden die Geraden nach (2.21) mit

$$G_1(SO, NO) \qquad G_2(NO, NW) \qquad G_3(SW, NW) \tag{5.10}$$

gebildet. Die Gerade $G_4(SO, SW)$ wird nicht berücksichtigt, da diese nur geschnitten werden kann, wenn vorher eine der anderen Geraden geschnitten wird. Für $G'(P_1', P_2')$ müssen die Koordinaten für dt berechnet werden. Unter der Annahme einer konstanten Geschwindigkeit berechnen sich die Koordinaten zum Zeitpunkt dt mit

$$\begin{pmatrix} x_{dtr} \\ z_{dtr} \end{pmatrix} = \begin{pmatrix} x_r \\ z_r \end{pmatrix} + \begin{pmatrix} \dot{x_r} \\ \dot{z_r} \end{pmatrix} \cdot dt. \tag{5.11}$$

Für die Objekte ergeben sich die Geraden $G'(P_1', P_2')$ zu

$$G_1'((x_r, z_r), (x_{dtr}, z_{dtr})) \qquad G_2'((x_r + w_r, z_r), (x_{dtr} + w_r, z_{dtr})) \tag{5.12}$$

Jetzt wird für jede Gerade G der Schnittpunkt mit jeder Geraden G' berechnet. Liegt ein Schnittpunkt innerhalb der ROI und tritt innerhalb von dt auf, wird die Meldung *Potentielle Gefahr* generiert.
Bevor die Einstufung *Keine Gefahr* vorgenommen werden kann, muss ein Sonderfall überprüft werden. Schneidet die Bounding Box ROI_z innerhalb von dt und die Bounding Box ist breiter als die ROI, muss überprüft werden, ob die ROI zwischen den Eckpunkten liegt oder ob sich die ROI seitliche neben dem Objekt befindet. Befindet sich das Objekt seitlich, wird die Gefahr auf *Keine Gefahr* eingestuft, sonst auf *Potentielle Gefahr*.

5.2.4 Objektverfolgung

Aus Abschnitt 5.2.3 geht hervor, dass die Kollisionsvorhersage den Zustandsvektor $\mathbf{x}_{predict}$ aus (5.8) benötigt, um eine Kollision vorhersagen zu können. Die Objektverfolgung soll genau diese Informationen bereitstellen, muss also diese Informationen berechnen.
Der in Abschnitt 3.3.2 vorgestellte SORT-Algorithmus bietet neben der sehr geringen Ausführungszeit den Vorteil, dass ein Kalman-Filter verwendet wird, um die Position

von erkannten Objekten für das nachfolgende Bild zu berechnen. Dafür wird der in (3.3) eingeführte Zustandsvektor $\mathbf{x}_{sort}$ verwendet. Der Vorteil, dass der Algorithmus zum einen die Objektverfolgung und zum anderen die Schätzung der Positionsänderung beinhaltet, wird im weiteren Verlauf dieser Arbeit genutzt. Der Zustandsvektor und das Zustandsraummodell müssen dafür angepasst werden, sodass anstelle der Bildkoordinaten direkt die Position und die Positionsänderung des Objektes in Roboterkoordinaten geschätzt wird. Für die Anpassung werden die Bildkoordinaten (u, v) für die Position und $(\dot{u}, \dot{v})$ für die Positionsänderung durch die Roboterkoordinaten (x_r, y_r) und $(\dot{x_r}, \dot{y_r})$ ersetzt. Die Zustände $(s, r, \dot{s})$ werden weiterhin verwendet, jedoch über die Roboterkoordinaten berechnet. Die noch fehlenden Zustände $(z_r, \dot{z_r})$ werden in dem Zustandsvektor ergänzt. Der resultierende Zustandsvektor ergibt sich zu

$$\mathbf{x}_{track} = \begin{pmatrix} x_r & y_r & s & r & z_r & \dot{x_r} & \dot{y_r} & \dot{s} & \dot{z_r} \end{pmatrix}^T. \tag{5.13}$$

Die Zustände für die Position berechnet der SORT-Algorithmus aus der Bounding Box, die benutzt wird, um die Eigenschaften der Objekte zu beschreiben. Die übergebene Bounding Box aus (3.2) wird um den Zustand z_r erweitert und die Angaben jeweils in Roboterkoordinaten überführt. Daraus ergibt sich der Zustandsvektor zur Beschreibung der Bounding Box zu

$$\mathbf{x}_{bb-track} = \begin{pmatrix} x_r & y_r & w_r & h_r & z_r \end{pmatrix}^T. \tag{5.14}$$

5.2.5 Objekterkennung

Die Objekterkennung muss den in (5.14) definierten Zustandsvektor für die Objektverfolgung bereitstellen. Die Genauigkeit des SORT-Algorithmus hängt direkt mit der Genauigkeit der Objekterkennung zusammen. Aus diesem Grund werden zwei unterschiedliche Objekterkennungsalgorithmen umgesetzt und anschließend die Ergebnisse miteinander verglichen. Anschließend wird ein Algorithmus ausgewählt, der in das System integriert ist.
Bei den Algorithmen, die miteinander verglichen werden, handelt es sich zum einen um den in [29] vorgestellten Algorithmus zur Hinderniserkennung in unwegsamem Gelände, welcher die v-Disparity Map verwendet, und zum anderen um den in [27] vorgestellten Algorithmus zur Erkennung von Hindernissen im Straßenverkehr, welcher die u-Disparity Map verwendet. Beide Algorithmen benötigen sowohl die Disparity Map als auch die Depth Map um den Zustandsvektor für die Objektverfolgung zu berechnen.

5.2.6 Sensordatenverarbeitung

Bei der Verarbeitung der Sensordaten müssen die in Kapitel 4.5.2 gestellten Anforderungen an die Verarbeitung von personenbezogenen Daten berücksichtigt werden. Da es nicht möglich ist, aus den 3D-Bilddaten relevante personenbezogene Daten zu gewinnen, werden für die zu verwendenden Algorithmen lediglich die 3D-Bilddaten verwendet. Die RGB-Bilddaten werden ausschließlich für die Validierung der Algorithmen genutzt. Die Personen, die sich in diesen Aufnahmen befinden, nehmen bewusst an der Erprobung des Forschungsvorhabens teil, weshalb nach der Einschätzung von [14] eine Verarbeitung der personenbezogenen Daten dieser Probanden zulässig ist.
Es ist nicht möglich die Depth Map zusammen mit der Disparity Map aus der Kamera auszulesen. Es muss folglich eine Map ausgelesen und die andere aus ihr berechnet werden.
Die Depth Map wird in dem Datenformat unsigned integer mit 16-Bit (uint16) und einem Umrechnungsfaktor in Metern ausgelesen. Die Disparity Map wird im 32-bit float (float32) Datenformat ausgelesen. Die Faktoren zum Umrechnen in eine geeignete Einheit müssen aus verschiedene Parametern ermittelt werden. Da die Disparity Map für die weitere Verwendung auf den Datentyp uint8 gemappt wird, wie in Abschnitt 2.2.2 vorgestellt, bietet es sich an, die Disparity Map aus der Depth Map zu berechnen. Die Disparity Map wird direkt aus der Depth Map mit einem vorgegebenen Wertebereich, der dem Arbeitsbereich der Kamera entlang der z-Koordinate entspricht, berechnet.

6 Design

In diesem Kapitel wird beschrieben, wie die Hard- und die Software designt werden.

6.1 Hardwaredesign

In diesem Abschnitt wird das Design der Hardware vorgestellt. Zum einen wird das Design des Roboters vorgestellt und zum anderen das Design des Teststandes, welcher zum Aufnehmen von Referenzvideos verwendet wird.

6.1.1 Roboter-Hardware

Als Referenzfahrzeug wird das in Abschnitt 2.5 vorgestellte Modell des Mars Rovers, ExoMy, verwendet. Der Kopf des Roboters, in dem sich die Monokamera befindet, wird durch die Intel Realsense D455, welche in Abschnitt 5.2.2 ausgewählt wird, ausgetauscht. Die Kamera wird mit einer 3D-gedruckten Adapterplatte auf dem Turm des Roboters montiert. Der modifizierte Roboter ist in Abbildung 6.1 dargestellt.

6.1.2 Hardware-Teststand

Um die Kollisionsvermeidung zu testen, wird ein Teststand entworfen, auf dem sich die Kamera oder Objekte mit einer konstanten Geschwindigkeit bewegen können. Der Teststand soll dafür verwendet werden, um die verwendeten Algorithmen bewerten zu können und die Genauigkeit der Ergebnisse interpretieren zu können.
Um die Bewegungen zu erzeugen, wird ein Schienensystem entworfen, auf dem sich zwei Gleiter nebeneinander zusammen bewegen können. Der Teststand ist in Abbildung 6.2 dargestellt. Die Gleiter werden über eine Plexiglasplatte miteinander verbunden. Die Plexiglasplatte dient als Plattform, auf der die Kamera oder Objekte montiert werden, um

Abbildung 6.1: ExoMy mit Stereokamera

mit konstanter Geschwindigkeit bewegt zu werden. An der Plattform wird ein Zahnriemen befestigt, der von einem Schrittmotor angetrieben wird. Der Motor ist an einer Seite des Teststandes mit einer 3D-gedruckten Motorhalterung befestigt. Auf der anderen Seite wird eine Umlenkrolle angebracht, die den Zahnriemen in der Spur hält. Die Ansteuerung des Motors wird mit einem Arduino und einem Schrittmotortreiber umgesetzt. Die Steuerung erfolgt mit einem Controller, der Signale an den Analogeingang des Arduinos sendet.

6.2 Softwaredesign

Das in Abschnitt 5.2.2 vorgestellte Konzept der Software wird in diesem Abschnitt spezifiziert. Der in Abbildung 5.1 skizzierte Ablauf der Kollisionsvermeidung dient als Grundlage für das weitere Vorgehen. Zu Beginn wird festgelegt, wie die einzelnen Aktivitäten des Systems mit ROS2 umgesetzt werden und wie modular die Software aufgebaut wird. Anschließend wird der Ablauf der einzelnen Komponenten separat dargestellt.

Abbildung 6.2: Hardware Teststand mit montierter Kamera

6.2.1 Design in ROS2

Bei der Architektur der Software werden die spezifischen Anforderungen an die Software berücksichtigt. Um die Software leicht erweiterbar zu gestalten und einzelne Komponenten in andere Projekte einzubinden, werden die in Abschnitt 5.2.2 hergeleiteten Arbeitsschritte in separate Nodes unterteilt. Jede Node ist ein eigener Prozess, der über Topics mit den anderen Nodes kommuniziert und berechnete Werte einliest oder übergibt. Aus Abschnitt 5.2 geht hervor, welche Daten in den jeweiligen Arbeitsschritten benötigt werden und welche sie für die anderen Arbeitsschritte bereitstellen.
Bei der Sensordatenverarbeitung werden große Matrizen erzeugt, die an die Objekterkennung übergeben werden müssen. Da die Arbeitsschritte direkt nacheinander ausgeführt werden und um den Datenfluss zwischen den Nodes gering zu halten, werden die Sensordatenverarbeitung und Objekterkennung in einer Node zusammengefasst. Die sich für ROS2 ergebende Architektur der Kollisionsvermeidung ist in Abbildung 6.3 dargestellt. Die blauen Boxen geben an, um welche Node es sich handelt. Die grauen Boxen in den Nodes enthalten die Funktionen, die in den Nodes abgearbeitet werden. In den grünen Boxen sind die Topics hinterlegt, welche die Nodes verwenden, um Messages zu verschi-

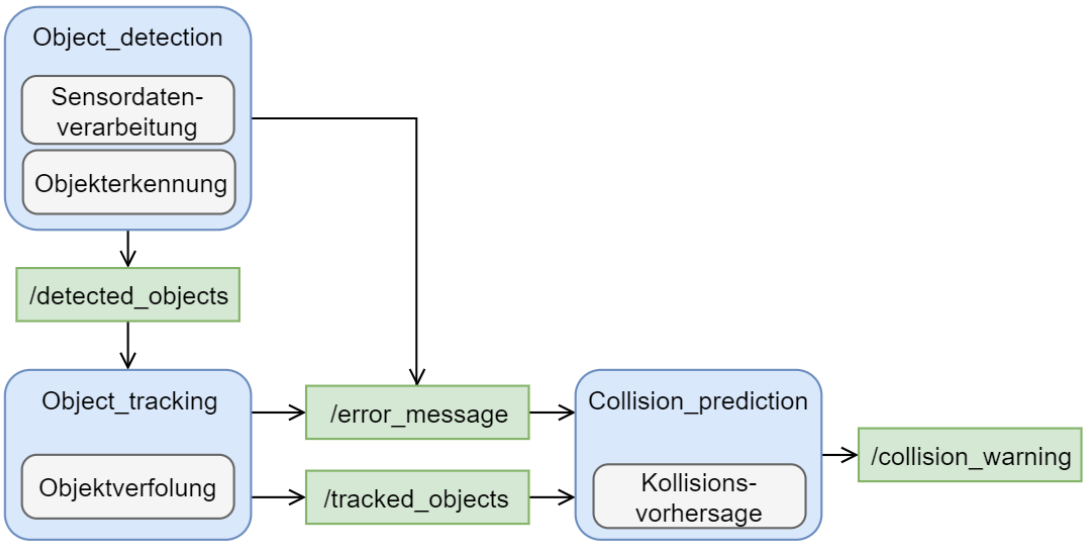


Abbildung 6.3: Programm-Architektur in ROS2

cken. Die Pfeile, die in die Topic reingehen, geben an, dass die Node Messages an dieses Topic sendet und Pfeile, die von der Topic weg zeigen, geben an, dass eine Node das Topic abonniert. Von der *Collision_prediction* wird eine Topic generiert, die anderen Systemen, wie beispielsweise der Steuerung, bereitgestellt wird, um die Kollisionswarnung zu interpretieren und einer möglichen Kollision zu vorzubeugen.
Jede Node wird in einem separaten Package implementiert, das ermöglicht es, die Nodes möglichst separat voneinander zu entwickeln und den Aufbau des Systems modular zu halten. Der Aufbau der Messages wird in ein weiteres Package ausgelagert, welches von den einzelnen Nodes eingebunden werden kann.

6.2.2 Aufbau der Nodes

Der übergeordnete Aufbau der Nodes wird gleich gestaltet. Die Nodes werden so aufgebaut, dass sie beim Starten der Software initialisiert werden und anschließend ihre Funktionen eventbasiert zyklisch ausführen. Diese Events können eintreffende Messages oder ablaufende Timer sein. Jede Node verfügt über einen Timer *error_timer*, der überwacht, ob die Nodes Arbeiten oder ein Fehler aufgetreten ist. Die Zeit des *error_timer* t_{max} wird so gewählt, dass Überschreitungen von Ausführungszeiten oder einzelne verlorene Frames nicht zu einem Auslösen des Timers führen. Wartet eine Node beispielsweise auf die Message eines Topics, läuft parallel der *error_timer* runter. Läuft der Timer ab,

wird eine Fehlermeldung gesendet. Wird die Message vor dem Ablauf von *error_timer* eingelesen, liegt kein Fehlerfall vor und der *error_timer* wird zurückgesetzt.
Beim Initialisieren wird festgelegt, an welche Topics die Producer Messages senden und von welchen Topics die Consumer Messages empfangen. Die Parameter für jede Node werden mit Standardwerten hinterlegt, welche jedoch beim Starten der Nodes verändert werden können. Im Folgenden wird auf den Ablauf der einzelnen Nodes separat eingegangen.

Object_detection

Die Node Object_detection enthält die Komponenten Sensorverarbeitung und Objekterkennung. Der Ablauf ist in Abbildung 6.4 dargestellt. Der zyklische Ablauf der Node

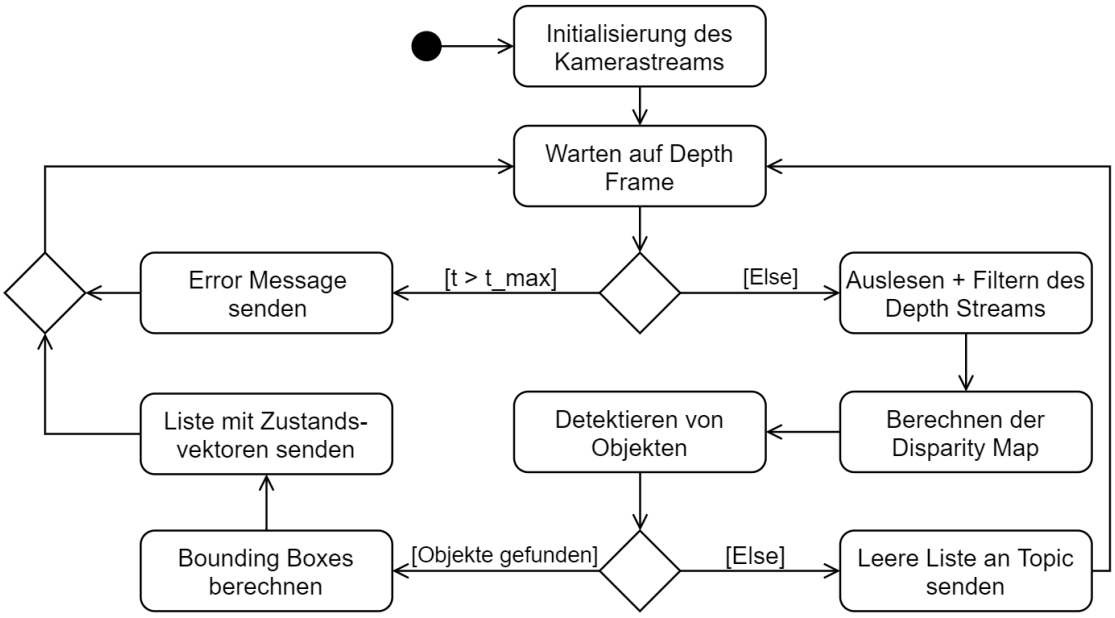


Abbildung 6.4: Ablauf der Objekterkennung

startet mit der Sensordatenverarbeitung, welche die benötigten Informationen für die Objekterkennung aufbereitet. Zuerst wird der Datenstrom mit den Tiefeninformationen von der Kamera eingelesen. Es müssen dabei neue Daten innerhalb einer vorgegebenen Zeit t_{max} bereitgestellt werden, um einen sicheren Ablauf des Systems zu garantieren. Wird t_{max} überschritten, wird eine Fehlermeldung an das Topic */error_message* gesendet. Um den zyklischen Ablauf aufrecht zu erhalten, wird anschließend auf den nächsten Daten-

strom gewartet. Wird innerhalb von t_{max} ein Datenstrom ausgelesen, wird der *error_timer* zurückgesetzt und der Datenstrom gefiltert. Die Entfernungswerte, die außerhalb des Arbeitsbereichs liegen, werden herausgefiltert. Mit den gefilterten Tiefeninformationen wird nun die Disparity Map berechnet und anschließend der Objekterkennungsalgorithmus aktiviert. Liefert der Algorithmus eine Liste mit Objekten zurück, wird anhand der Depth Map und Position der jeweilige Zustandsvektor für die Objektverfolgung berechnet und die Liste mit Zustandsvektoren an das Topic */detected_objects* gesendet. Werden von dem Objekterkennungsalgorithmus keine Objekte erkannt, wird eine leere Liste an */detected_objects* gesendet. Nach dem Senden der Listen beginnt der zyklische Vorgang wieder beim Warten auf den nachfolgenden Datenstrom.

Object_tracking

Die Objektverfolgung wird in der Node *Object_tracking* umgesetzt. Die Node wird ebenfalls nach dem Starten initialisiert und anschließend beginnt der zyklische Funktionsablauf, welcher in Abbildung 6.5 dargestellt ist. Die Objektverfolgung wartet auf Messages,

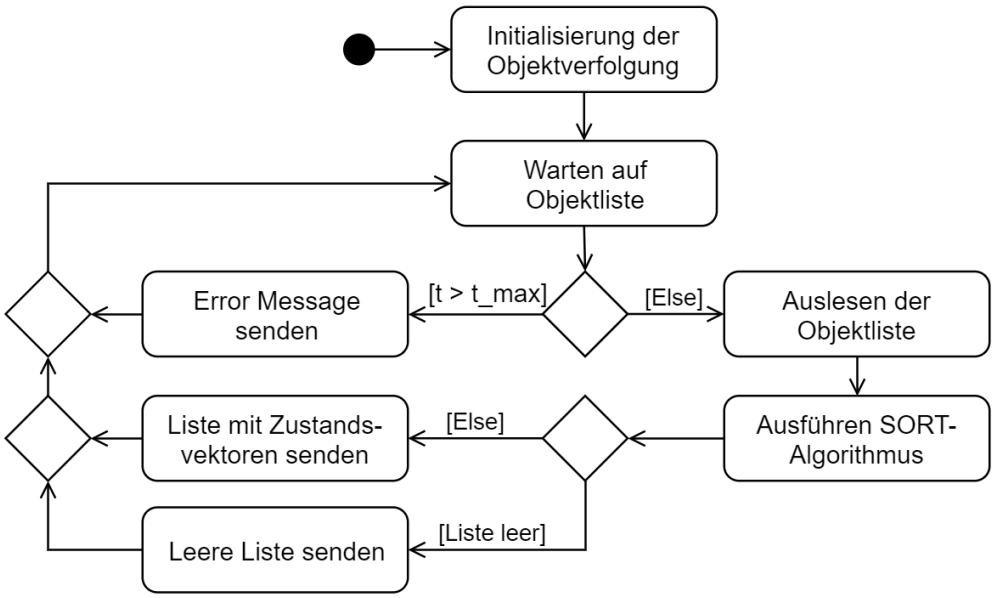


Abbildung 6.5: Ablauf der Objektverfolgung

die an */detected_objects* gesendet werden und Listen von Zustandsvektoren enthalten. Werden keine Objekte erkannt, werden leere Listen übergeben und von der Objektverfolgung eingelesen. Wird länger als t_{max} gewartet, wird der SORT-Algorithmus zurück-

gesetzt. Das bedeutet, alle erkannten Objekte werden gelöscht und eine Message wird an das Topic */error_message* gesendet. Liegt innerhalb von t_{max} eine Objektliste vor, wird diese Liste eingelesen, der *error_timer* zurückgesetzt und der SORT-Algorithmus ausgeführt. Sind keine Objekte vorhanden, die verfolgt werden können, wird eine leere Liste an das Topic */tracked_objects* gesendet. Andernfalls werden die Zustandsvektoren der verfolgten Objekte in einer Liste an */tracked_objects* gesendet. Anschließend beginnt der zyklische Vorgang wieder beim Warten auf eine neue Objektliste.

Collision_prediction

Die Node Collision_prediction enthält die Kollisionsvorhersage. Der Ablauf ist in Abbildung 6.6 dargestellt. Neben dem *error_timer* wird in dieser Node ein weiterer Timer

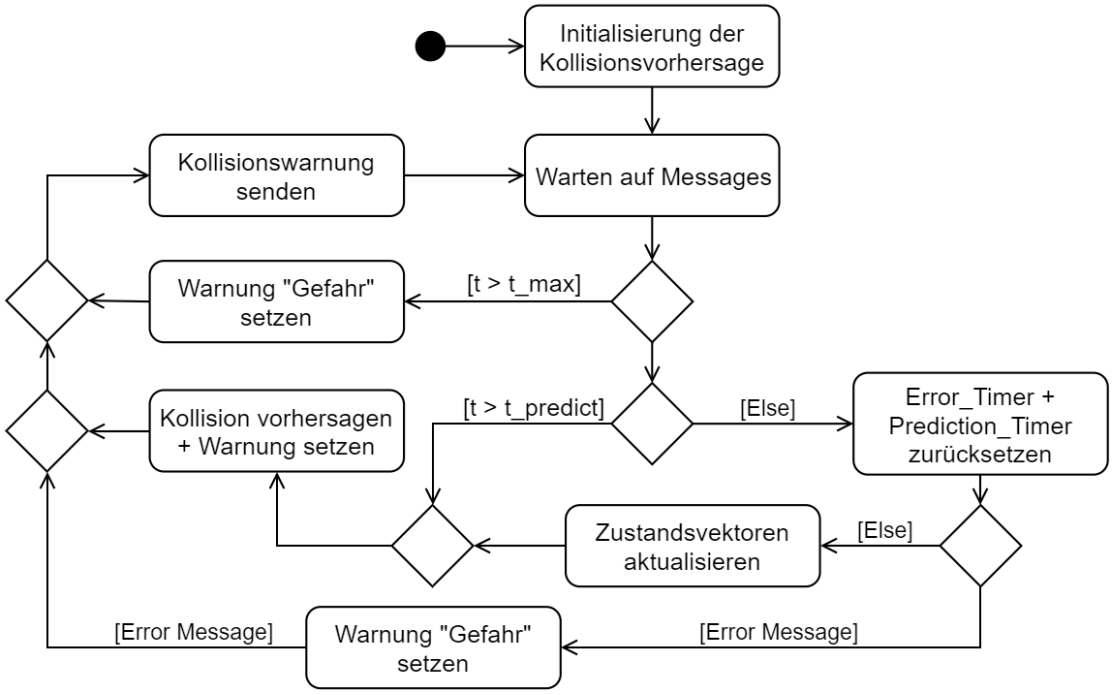


Abbildung 6.6: Ablauf der Kollisionsvorhersage

prediction_timer verwendet. Der *prediction_timer* enthält eine kleinere Zeitkonstante $t_{predict}$ als der *error_timer* und sorgt dafür, dass die Wahrscheinlichkeit einer Kollision innerhalb von $t_{predict}$ ausgeführt wird, auch wenn beispielsweise einzelne Frames verloren gehen.

Nach dem Starten der Node wird diese direkt initialisiert und anschließend der zyklische Ablauf gestartet. Die Kollisionsvorhersage wartet auf Messages von den Topics */error_message* und */tracked_objects*. Es muss innerhalb von t_{max} eine Message von einer der beiden Topics empfangen werden, ansonsten wird die Kollisionswarnung *Gefahr* an das Topic */collision_warning* gesendet. Wird eine Message vom Topic */error_message* eingelesen, wird direkt eine Kollisionswarnung an */collision_warning* gesendet und die Timer *error_timer* und *prediction_timer* zurückgesetzt. Ist die Message von dem Topic */tracked_objects*, wird die Objektliste überschrieben, die Timer *error_timer* und *prediction_timer* zurückgesetzt und die Wahrscheinlichkeit einer Kollision berechnet. Anschließend wird die Einstufung der Gefahr an */collision_warning* gesendet. Wird innerhalb von $t_{predict}$ keine neue Message eingelesen, wird die Wahrscheinlichkeit einer Kollision mit den alten Objektzuständen durchgeführt, die Einstufung der Gefahr vorgenommen und eine Kollisionswarnung gesendet. Nach dem Senden der Kollisionswarnung, wird auf neue Messages gewartet.

7 Umsetzung der Module

In diesem Kapitel wird die Umsetzung der einzelnen Module aus dem Konzept vorgestellt. Die Entwicklung wird in mehrere Schritte unterteilt. Zu Beginn werden Parameter des Systems ermittelt. Anschließend werden die Komponenten Sensordatenverarbeitung, Objekterkennung, Objektverfolgung und Kollisionsvorhersage umgesetzt und ein Algorithmus für die Objekterkennung ausgewählt.

7.1 Ermittlung der Roboterparameter

Roboterkorrdinatensystem: Die Berechnung der Position von erkannten Objekten und die Kollisionsvorhersage aus den Daten erfolgt in dem Roboterkoordinatensystem. Die mit der Kamera aufgenommenen Informationen müssen dafür in das Roboterkoordinatensystem nach dem in Abschnitt 2.1.2 vorgestellten Verfahren überführt werden. Das Koordinatensystem des Roboters ist fest mit dem Roboter verbunden. Die Anordnung ist in Abbildung 7.1 dargestellt. Der Koordinatenursprung liegt für die x_r-Koordinate in

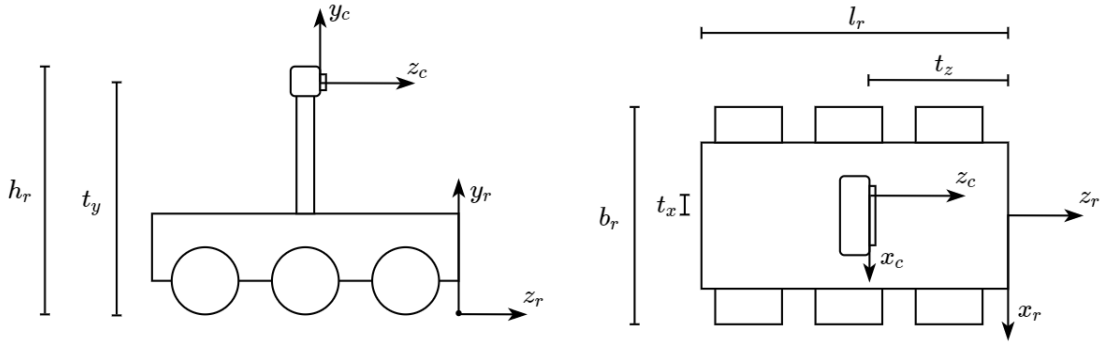


Abbildung 7.1: Anordnung der Koordinatensysteme

der Mitte des Roboters. Der Ursprung für die y_r-Koordinate wird auf dem unteren Ende der Reifen, also dem Boden festgelegt. Für die z_r-Koordinate wird der Ursprung auf den nördlichsten Punkte des Roboters gelegt.

Koordinatentransformation: Um die Koordinatentransformation auszuführen, müssen die Rotationsmatrix $\mathbf{R}$ und der Verschiebungsvektor $\mathbf{t}$ ermittelt werden. Die Verschiebung $\mathbf{t}$ der Koordinatensysteme ist in Abbildung 7.1 mit dargestellt. Die Verschiebung in x-Richtung ist in der Abbildung mit t_x gekennzeichnet. Die Kamera ist auf dem Roboter so angeordnet, dass der Mittelpunkt der Kamera auf dem Mittelpunkt des Roboters liegt, also bei $x_r = 0$. Der Koordinatenursprung der verwendeten Kamera liegt mittig in der linken Linse, der Stereokamera. Für t_x ergibt sich somit der Abstand des Kamerazentrums zur linken Kameralinse, welcher $t_x = 47,5$ mm beträgt. Für y_c befindet sich der Koordinatenursprung ebenfalls mittig in der linken Kameralinse. Für t_y ergibt sich somit die Summe der Höhe der Kamerahalterung h_c und der Abstand von der Kameramontage bis zum Zentrum der linken Linse mit $t_y = 395 \text{ mm} + 14,5 \text{ mm} = 409,5$ mm. Der Koordinatenursprung für z_c befindet sich innerhalb des Gehäuses der Kamera. Ausgehend von der Glasscheibe vor der Kameralinse muss ein Offset von $4,55$ mm berücksichtigt werden. Für t_z ergibt sich als Summe aus dem Abstand der Roboter- und Kamerafront und dem Kameraoffset $t_z = 110 \text{ mm} + 4,55 \text{ mm} = 114,55$ mm.
Die Kamera wird so auf dem Roboter angeordnet, dass die Koordinatensysteme für jede Achse parallel verlaufen. Damit ist die Kamera entlang der Fahrtrichtung des Roboters ausgerichtet und es ist möglich die größte Entfernung entlang z_r zu erreichen. Für $\mathbf{R}$ ergibt sich die Einheitsmatrix. Eingesetzt in (2.8) ergibt sich für die Transformationsmatrix mit auf Millimeter gerundeten Werten.

$$\begin{pmatrix} x'_r \\ y'_r \\ z'_r \\ w_r \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & -0,048 \\ 0 & 1 & 0 & 0,410 \\ 0 & 0 & 1 & -0,115 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x'_c \\ y'_c \\ z'_c \\ w_c \end{pmatrix} \tag{7.1}$$

Roboterabmessungen: Die Abmessungen des Roboters ergeben sich aus den in Abbildung 7.1 dargestellten Größen h_r für die Höhe des Roboters, b_r für die Breite des Roboters und l_r für die Länge des Roboters. Die Größen werden am Roboter gemessen und ergeben sich zu $h_r = 420$ mm, $b_r = 290$ mm und $l_r = 400$ mm

Region of Interest: Die ROI legt fest, wie nah Objekte an den Roboter herankommen dürfen, bevor eine Meldung *Gefahr* oder *potentielle Gefahr* generiert wird. Für eine Auflösung von (240×424) ist die minimale Entfernung, die die Kamera aufnehmen kann, mit $z_{min} = 0,180$ m im Datenblatt angegeben [17]. Der optimale Arbeitsbereich beginnt laut

Herstellerangaben jedoch ab $0,6$ m. Um sicherzustellen, dass Objekte innerhalb der ROI erkannt werden und der Roboter manövrierfähig ist, wird $ROI_{z\_max} = 0,6$ m festgelegt. Damit besteht die Möglichkeit auch Objekte zu erkennen, die sich in der ROI befinden. Für die restlichen Grenzen der ROI wird jeweils die Roboterbreite, Roboterlänge oder Roboterhöhe als Sicherheitsabstand festgelegt. Für die hintere Grenze ergibt sich $ROI_{z\_min} = -0,400$ m $- 0,400$ m $= -0,82$ m. Für die ROI entlang der x-Achse ergibt sich ein Bereich von $-0,630$ m $\rightarrow 0,630$ m. Objekte, unter die der Roboter durchfahren kann, sind ebenfalls außerhalb der ROI. Die Objekte müssen dabei die doppelte Höhe des Roboter aufweisen. Daraus ergibt sich $ROI_{y\_max} = 0,840$ m

7.2 Kollisionsvorhersage

Wie in Abschnitt 5.2.3 beschrieben, wird die Kollisionsvorhersage in zwei Schritten umgesetzt. Die Umsetzung dieser Schritte wird im folgenden genauer erläutert.

Objekt befindet sich in der ROI

Die Logik für die Überprüfung, ob sich ein Objekt innerhalb der ROI befindet, wird nach dem Vorgehen in Teilabschnitt 5.2.3 in Algorithmus 1 umgesetzt. Die Überprüfung, ob das Objekt in der ROI ist, wird für jede Achse separat durchgeführt. Befindet sich das Objekt für jede Achse in der ROI, wird *true* zurückgegeben, ansonsten *false*. Für die x-Achse wird überprüft, ob der westlichste Punkt des Objektes $x_r + w_r$ östlich von der ROI liegt oder ob der östlichste Punkt des Objekts x_r westlich von der ROI liegt. Trifft eine dieser Bedingungen zu, befindet sich das Objekt für die x-Koordinate außerhalb der ROI, Zeile 2, andernfalls befindet sich das Objekt innerhalb von ROI_x, Zeile 4. Für die y-Achse wird überprüft, ob sich der niedrigste Punkt des Objektes y_r oberhalb von ROI_y befindet. Ist das der Fall, befindet sich das Objekt außerhalb der ROI, Zeile 5. Für die z-Koordinate wird überprüft, ob sich das Objekt nördlich oder südlich der ROI befindet, Zeile 7. Befindet sich das Objekt südlich oder nördlich wird *false* zurück gegeben, Zeile 8. Sonst befindet sich das Objekt für jede Achse in der ROI und es wird *true* zurückgegeben, Zeile 10.

Algorithm 1 Object in ROI

1: **if** $(x_r + w_r < ROI_{East})$ **or** $(x_r > ROI_{West})$ **then**
2: **return** $false$
3: **end if**
4: **if** $y_r > ROI_y$ **then**
5: **return** $false$
6: **end if**
7: **if** $(z_r > ROI_{North})$ **or** $(z_r < ROI_{South})$ **then**
8: **return** $false$
9: **else**
10: **return** $true$
11: **end if**

Objekt bewegt sich in die ROI

Die Überprüfung, ob sich ein Objekt innerhalb von *dt* in die ROI bewegt, erfolgt nach dem in Teilabschnitt 5.2.3 dargestellten Konzept und ist in Algorithmus 2 umgesetzt. Der Algorithmus gibt den Wert *true* zurück, wenn sich das Objekt in die ROI bewegt, sonst *false*. Befindet sich das Objekt zum Zeitpunkt der Bildaufnahme und zum Ende des Horizonts außerhalb von ROI_y, wird *false* zurückgegeben, Zeile 2. Andernfalls befindet sich das Objekt teilweise oder die ganze Zeit innnerhalb von ROI_y und es werden alle möglichen Schnittpunkte der ROI mit den Eckpunkten der Bounding Box für die x, z-Ebene berechnet. Die Berechnung gibt eine Liste mit den Koordinaten der Schnittpunkte zurück, Zeile 4. Für die Faktoren t und t' wird paarweise für jeden berechneten Schnittpunkt überprüft, ob sie zwischen null und eins liegen. Ist das für ein Paar der Fall, schneiden sich diese Geraden innerhalb der ROI und innerhalb von *dt* und es wird *true* zurückgegeben, Zeile 6. Schneiden sich die Punkte alle außerhalb der ROI, muss für den Sonderfall überprüft werden, ob das Objekt die ROI nördlich innerhalb von *dt* schneidet und dabei ein Punkt östlich und ein Punkt westlich schneidet. Trifft dieser Fall zu, bewegt sich das Objekt ebenfalls in die ROI. Es wird *true* zurückgegeben, Zeile 9. Trifft keine Überprüfung zu wird *false* zurückgegeben.

7.3 Objektverfolgung

Wie in Abschnitt 5.2.4 beschrieben, wird der SORT-Algorithmus von fünf auf sieben Zustände erweitert. Entsprechend der neuen Zustände muss auch das ZRM für das Kalman-

Algorithm 2 Object moves in ROI

```
if (y_r > ROI_y)  and  (y_dtr > ROI_y) then
    return false
end if
(S, t, t') = calculate_all_intersections(G, G')
if any 0 ≤ t ≤ 1  and  0 ≤ t' ≤ t' then
    return true
end if
if S(G_2, G'_1) < ROI_East  and  S(G_2, G'_2) > ROI_West and 0 < t'(G_2, G'_1) < 1 then
    return true
end if
return false
```

Filter angepasst werden. Für die Zustandsgleichung ergibt sich

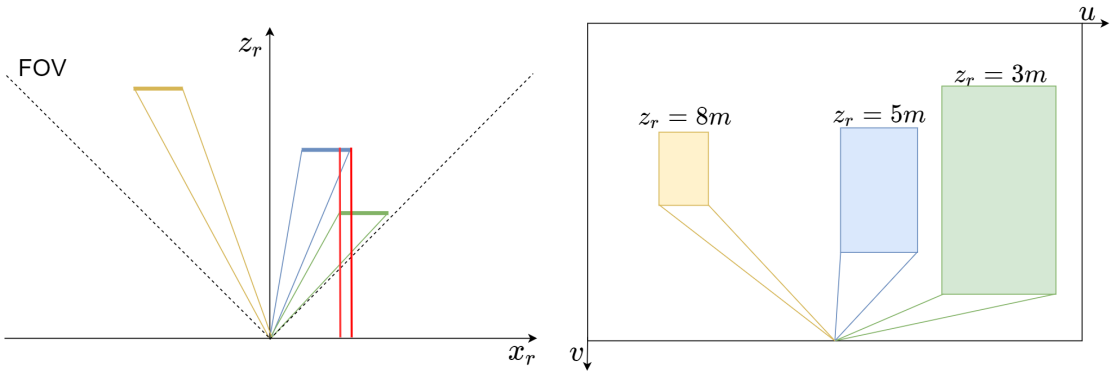
$$\mathbf{x}_{track}(k+1) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_r(k) \\ y_r(k) \\ s_r(k) \\ r_r(k) \\ z_r(k) \\ \dot{x_r}(k) \\ \dot{y_r}(k) \\ \dot{s_r}(k) \\ \dot{z_r}(k) \end{pmatrix} \tag{7.2}$$

und für die Ausgabegleichung ergibt sich

$$\mathbf{y}(k) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_r(k) \\ y_r(k) \\ s(k) \\ r(k) \\ z_r(k) \\ \dot{x_r}(k) \\ \dot{y_r}(k) \\ \dot{s_r}(k) \\ \dot{z_r}(k) \end{pmatrix}. \tag{7.3}$$

Erste Tests haben gezeigt, dass die Objektverfolgung mit Roboterkoordinaten Objekte nicht so genau verfolgen kann wie mit Bildkoordinaten. In Abbildung 7.2a sind exempla-

risch drei gleich große Objekte im Roboterkoordinatensystem für die x, z-Koordinaten und in Abbildung 7.2b die Abbildung der Objekte im Bildkoordinatensystem skizziert. Es ist zu erkennen, dass sich das grüne und blaue Objekt im Roboterkoordinatensystem



(a) Position der Objekte im Roboterkoordinatensystem

(b) Abbildung der Objekte im Bildkoordinatensystem

Abbildung 7.2: Darstellung von erkannten Objekten

entlang der x-Achse schneiden, mit den roten senkrechten Strichen gekennzeichnet. Aufgrund der größeren Entfernung des blauen Objektes, ist im Bildkoordinatensystem jedoch deutlich zu erkennen, dass sich die Bounding Boxes nicht schneiden. Wird für die Objekte in der Roboterkoordinatendarstellung die IOU berechnet, so ergibt sich $0 \leq \text{IOU} \leq 1$, obwohl es sich eindeutig um zwei verschiedene Objekte handelt. Wird die Überschneidung zu groß, kann es zu Fehlzuordnungen kommen und die Ergebnisse der Objektverfolgung sind für die Kollisionsvorhersage nicht mehr zuverlässig.
Um die Objekte in der Roboterkoordinatendarstellung unterscheiden zu können, wird die Entfernung der Objekte bei der Berechnung der IOU berücksichtigt. Die in Teilabschnitt 5.2.5 vorgestellten Algorithmen zur Objekterkennung berechnen nur die Breite der Objekte entlang der x-Achse, jedoch nicht entlang der z-Achse. Aus diesem Grund ist es nicht möglich, die Berechnung der IOU auf das Volumen der Objekte zu erweitern. Um die Entfernung trotzdem zu berücksichtigen, wird die in Abschnitt 4.5.2 getroffene Annahme, dass sich Objekte mit einer maximalen Geschwindigkeit von $v_{max} = 4,36\text{m/s}$ bewegen, aufgegriffen und bei der IOU Berechnung berücksichtigt.
Die Berechnung der IOU zwischen allen erkannten Objekten und den prädizierten Zuständen der Kalman-Filter wird wie zuvor berechnet. Anschließend werden die berechneten Werte der IOU auf null gesetzt, bei denen die Entfernung z zwischen den Bounding Boxes größer ist als die maximale Positionsänderung zwischen zwei aufgenommenen Bildern. Bewegen sich Objekte mit v_{max} und werden die Bilder mit 15Hz ausgelesen, be-

wegen sich die Objekte zwischen den Bildern maximal $0,29$m. Aufgrund der unebenen Objektoberflächen kann die Entfernung des gleichen Objektes zwischen zwei Bildern abweichen. Eine Referenzmessung von einer stehenden Person ergab eine Abweichung von $0,6$m. Unter der Berücksichtigung der Geschwindigkeit und den Abweichungen ergibt sich $z_{max} = 0.91$m.

7.4 Objekterkennung und Sensordatenverarbeitung

Wie bereits in der Konzeption vorgestellt wird, werden zwei verschiedene Algorithmen getestet, um die Objekterkennung umzusetzen. Die Ergebnisse werden nach der ersten Umsetzung miteinander verglichen und anschließend ein Verfahren ausgewählt, welches im weiteren Verlauf verwendet wird. Das ausgewählte Verfahren legt fest, welche Schritte bei der Sensordatenverarbeitung vorgenommen werden müssen. Die Algorithmen werden auf einem Windows PC mit einem 2,5GHz Prozessor umgesetzt und dort getestet

7.4.1 Berechnung der Disparity Map

Die Berechnung der Disparitätswerte erfolgt nach Umstellung von (2.10) zu

$$p = \frac{a \cdot b}{g}. \tag{7.4}$$

Die von der Kamera ausgelesenen Tiefeninformationen der Depth Map werden im uint16-Format abgespeichert. Mit einer Auflösung von 1 mm lassen sich Entfernungen von 0 m $\rightarrow$ $65,535$ m darstellen. Der Arbeitsbereich aus den Anforderungen wird auf $0,4$ m $\rightarrow$ $8,72$ m festgelegt. Um die Reichweite der Kamera auch für weiter entfernte Objekte auszudehnen, wird der Arbeitsbereich auf 10 m festgelegt.
Für die Kamera sind die Parameter $b = 1,93$ mm und $a = 95$ mm. Daraus ergeben sich Disparitäten im Bereich von $458,4$ μm $\rightarrow$ $18,3$ μm. Um die Zahlenwerte für die Algorithmen handlicher zu gestalten, werden keine Disparitäten in Metern berechnet, sondern die Disparitätswerte auf einen Wertebereich festgelegt. Für die Berechnung der Disparity Map wird das Zahlenformat uint8 gewählt, da uint8 ein gängiges Zahlenformat in der Bildverarbeitung ist und Bibliotheken wie OpenCV Algorithmen bereitstellen, die nur im uint8 Format funktionieren [25]. Der Arbeitsbereich von $0,4$m-10m muss auf den Wertebereich von 1-255 übertragen werden. Allen Pixeln, die einen Wert außerhalb des Arbeitsbereiches aufweisen, wird der Wert 0 zugewiesen, um diese bei Berechnungen nicht

mit zu berücksichtigen. Die Gleichung wird nun um die Anpassung des Wertebereichs erweitert. Da die Parameter a und b konstant sind und in der Berechnung nur als Faktor auftreten, werden sie im Folgenden zu 1 gesetzt, wodurch sich für die Disparitäten

$$p = \frac{1}{g}. \tag{7.5}$$

ergibt. Das ist an dieser Stelle möglich, da die verwendeten Algorithmen zur Erkennung der Objekte nur die Disparitäten des vorgegebenen Wertebereichs miteinander vergleichen. Die genauen Informationen über die Entfernungen werden anschließend wieder aus der Depth Map gewonnen. Der Faktor F wird verwendet, um den Wertebereich der Disparity Map auf den Wertebereich von uint8 anzupassen. F wird mit

$$F = \frac{255 - 1}{\frac{1}{0{,}4} - \frac{1}{10}} \tag{7.6}$$

berechnet. Für die Disparity Map im uint8-Format werden die Disparitäten p mit

$$p = F \cdot \left(\frac{1}{g} - \frac{1}{10}\right) + 1. \tag{7.7}$$

berechnet. Der Verlauf der Disparitäten über den Arbeitsbereich ist in Abbildung 7.3 dargestellt. Es ist gut zu erkennen, dass die Auflösung in höheren Entfernungen abnimmt.

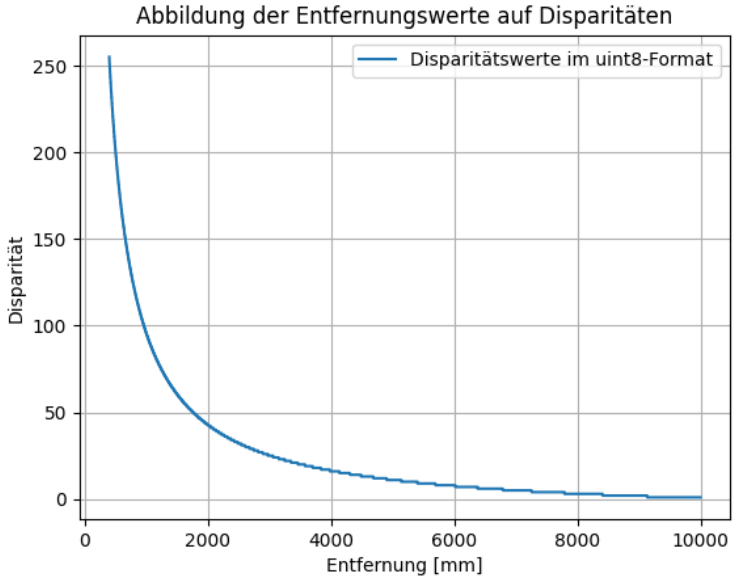


Abbildung 7.3: Abbildung der Entfernungen auf die Disparitäten

Für die Objekterkennung ist es dadurch für weit entfernte Objekte schwieriger eine genaue Abgrenzung durchzuführen. Die Berechnung der Position hingegen wird nicht so stark beeinträchtigt, da die Positionsberechnung anhand der Depth Map vorgenommen wird.

7.4.2 Objekterkennung mit der v-Disparity Map

Der in [29] vorgestellte Algorithmus verwendet die v-Disparity Map, um das Sichtfeld des Roboters zu analysieren. Für jede Zeile der v-Disparity Map wird versucht, die Disparität des Bodens zu finden. Alle Disparitätswerte, die größer sind als die des Bodens, werden Hindernissen zugeordnet. Um die Disparität des Bodens zu ermitteln, wird davon ausgegangen, dass der Boden für jede Zeile eine konstante Disparität mit kleinen Abweichungen aufweist. Dadurch liefert der Boden die meisten Disparitätswerte für jede Zeile. Es wird also eine zeilenweise Analyse durchgeführt und die in einer großen Häufung auftretenden Werte bilden die Disparität für den Boden. Die Werte des Bodens jeder Zeile werden in eine neue v-Disparity Map geschrieben, welche mit Mean Ground Depth (MGD) bezeichnet wird. Alle Pixel, die eine Disparität aufweisen, die größer ist als die des Bodens und ein dazugehöriger Schwellwert, gehören zu einem Objekt. Anschließend werden die Objekte nach der Größe gefiltert.

7.4.3 Objekterkennung mit der u-Disparity Map

Der in [27] vorgestellte Algorithmus verwendet die in Teilabschnitt 2.2.2 eingeführte u-Disparity Map um Objekte zu detektieren. Die Eigenschaft, dass Objekte im Gegensatz zum Boden eine Häufung von Disparitäten im spaltenweisen Histogramm aufweisen, wird genutzt, um die Objekte zu erkennen. Im ersten Schritt wird in [27] die u-Disparity Map berechnet und anschließend die Objekte aus ihr ermittelt. Dafür wird ein Kantenfilter auf die u-Disparity Map angewendet, wodurch die Disparitätswerte des Bodens und des Hintergrunds herausgefiltert werden und die Disparitäten der Objekte über bleiben. Für die Disparitäten, die sowohl in der gefilterten u-Disparity Map, als auch in der ursprünglichen Disparity Map auftreten, wird ein Binärbild erzeugt. Die Objekte erhalten den Wert 1 und der Rest den Wert 0. Anhand der Konturen können nun die Objekte erkannt werden.

7.4.4 Vergleich der Ergebnisse

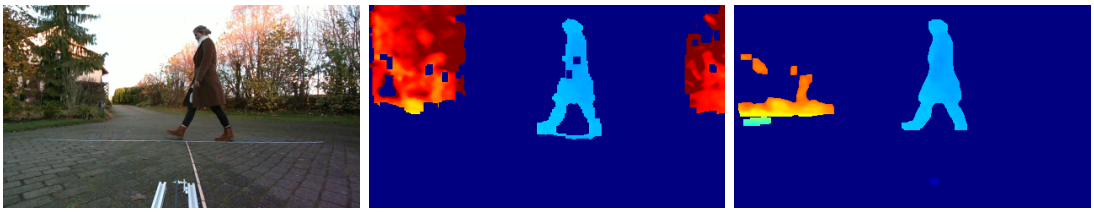
In diesem Teilabschnitt werdend die Ergebnisse der Algorithmen mit einander verglichen und ein Algorithmus ausgewählt, der im weiteren Verlauf der Arbeit eingesetzt wird.

Vergleich der Genauigkeit

Um die Genauigkeit bei der Objekterkennung zu analysieren, werden verschiedene Videos aufgenommen, welche mit beiden Verfahren getestet wird. Zudem wird ein RGB-Farbbild aufgenommen, mit dem anschließen die Validierung vorgenommen werden kann. In den Abbildungen 7.4 und 7.5 sind exemplarisch Ergebnisse der verwendeten Objekterkennungsalgorithmen gegenübergestellt.

(a) RGB-Bild (b) u-Disparity Map (c) v-Disparity Map

Abbildung 7.4: Ergebnisse der Objekterkennungsalgorithmen in hohem Gras



(a) RGB-Bild (b) u-Disparity Map (c) v-Disparity Map

Abbildung 7.5: Ergebnisse der Objekterkennungsalgorithmen auf einem gepflasterten Weg

In den Abbildungen 7.4a und 7.5a sind die mit der RGB-Kamera aufgenommenen Referenzbildbilder dargestellt. In 7.4b und In 7.5b ist jeweils das Ergebnis der Objekterkennung mit der u-Disparity Map und in 7.4c und 7.5c jeweils das Ergebnis der v-Disparity Map abgebildet. In den Ergebnisbildern sind die Hintergründe, die Bereiche in denen sich keine Objekte befinden, dunkelblau gekennzeichnet. Die Person, die sich in der jeweiligen Szene befindet, wird von beiden Algorithmen sicher erkannt. Bei größeren Entfernungen

nimmt die Genauigkeit des v-Disparity-Algorithmus jedoch ab, wodurch Büsche und Bäume, nicht sehr gut detektiert werden. Mit der u-Disparity hingegen werden auch weiter entfernte Objekte gut erkannt. Treten Unebenheiten im Boden auf, werden diese von dem v-Disparity-Algorithmus fälschlicherweise als Objekte erkannt. In [29] wird vorgeschlagen die fehlerhaft erkannten Objekte über den Neigungswinkel der Objekte herauszufiltern. Ein weiterer Effekt, der sich aus den Unebenheiten ergibt, ist jedoch, dass Teile der Beine abgeschnitten werden, wodurch sich die Fläche der Bounding Box verringert. Bei dem u-Disparity-Algorithmus wird der Boden in der Entfernung der erkannten Person fälschlicherweise als Objekt erkannt. Das stellt für den weiteren Verlauf jedoch kein Problem dar, da sich die Form der Bounding Box dadurch nur minimal verändert. Es fällt ebenfalls auf, dass der u-Disparity-Algorithmus bei Objekten, die keine ebene Fläche aufweisen, Teile, die zum Objekt gehören, nicht als Objekt erkennt. Dadurch entstehen kleine Löcher in den detektierten Bereichen. Diese Fehlklassifikation hat jedoch keinen Einfluss auf die Bounding Box, wodurch die Performance des SORT-Algorithmus nicht beeinträchtigt werden sollte.

Vergleich der Ausführungszeiten

Ein weiterer Aspekt, der bei der Auswahl berücksichtigt werden muss, ist die Ausführungszeit der jeweiligen Algorithmen. Für die Messung der Zeit wird nur die Ausführung der Algorithmen berücksichtigt. Auf einem herkömmlichen Windows Laptop ergibt sich für den v-Disparity Algorithmus eine Ausführungszeit von 10ms und für den u-Disparity-Algorithmus 27ms. Erste Tests auf dem Raspberry Pi haben gezeigt, dass die Ausführungszeiten ungefähr um das zehn-fache größer sind als auf dem Laptop. Die Verwendung von Numpy-Funktionen reicht nicht aus, um die Zykluszeiten weiter zu verringern. An dieser Stelle wird die Bibliothek Numba eingesetzt, die es ermöglicht Python Code in Maschinencode zu Übersetzung und damit die Ausführungszeiten von Funktionen zu verringern. Numba eignet sich dazu um Funktionen, die for-Schleifen beinhalten, oder numerische Funktionen ausführen zu beschleunigen. In dieser Arbeit wird Numba an zwei Stellen verwendet. Zum einen um die Berechnung der u-Disparity Map zu beschleunigen und zum anderen um die Disparitätswerte der Disparity Map mit denen der Kantengefilterten u-Disparity Map zu vergleichen.
Mit der Beschleunigung durch die Numba-Bibliothek ist es möglich die Objekterkennung mit der u-Disparity Map in 43 ms auszuführen.

Der v-Disparity Algorithmus ist damit mehr als doppelt so schnell wie der u-Disparity Algorithmus, es ist jedoch möglich mit beiden Algorithmen eine Echtzeitfähige Kollisionsvermeidung umzusetzen.

Auswahl des Algorithmus

Für das weitere Vorgehen wird der u-Disparity-Algorithmus verwendet, um die Objekterkennung umzusetzen. Der v-Disparity-Algorithmus ist zwar schneller, jedoch sind die Fehldetektionen ausschlaggebender. Vor allem die Anforderung an die Reichweite konnte mit dem v-Disparity Algorithmus nicht erfüllt werden.

8 Implementierung in ROS2

In diesem Abschnitt wird die Implementierung der einzelnen Nodes, die in 5.2 erläutert werden, vorgestellt. Die Implementierung der Algorithmen erfolgt in zwei Schritten. Im ersten Schritt werden dieNodes auf einem Windows PC nach der in 6.2 vorgegebenen Architektur umgesetzt. Anschließend werden die Packages auf der Zielhardware installiert und erprobt.

8.1 Umsetzung der Nodes

In diesem Abschnitt wird die Softwarearchitektur festgelegt, welche sich aus den Aktivitätsdiagrammen der einzelnen Nodes aus Abschnitt 6.2.2 definieren lässt. Für jede zu erstellende Nodes wird eine eigene Klasse erstellt, welche jeweils von der übergeordneten Klasse *rclpy.Node* erbt (Abb. 8.1). Der Aufbau der einzelnen Nodes erfolgt nach dem gleichen Schema. Für die Topics, an die Messages gesendet werden, werden Publisher erzeugt und für Topics, von denen Messages abonniert werden, werden Subscriptions erzeugt. Für das Überwachen der einzuhaltenden Zeiten werden Timer der Node-Klasse implementiert, die nach Ablauf einer vorgegebenen Zeit eine hinterlegte Funktion ausführen. Die Timer werden an bestimmten Stellen im Code zurückgesetzt, welche regelmäßig abgearbeitet werden, wenn die Nodes fehlerfrei funktionieren. Zusätzlich werden für die jeweiligen Funktionen zum Ausführen der Kernaufgabe, Objekterkennung, Objektverfolgung und Kollisionsvorhersage, die jeweiligen Parameter definiert. In den nachfolgenden Teilabschnitten wird die Implementierung der einzelnen Nodes genauer erläutert.

8.2 ObjectDetection

Für die Umsetzung der Logik des Aktivitätsdiagramms aus Abbildung 6.4 wird die Klasse *ObjectDetection* implementiert. Nach der Initialisierung der Nodes wird die Methode

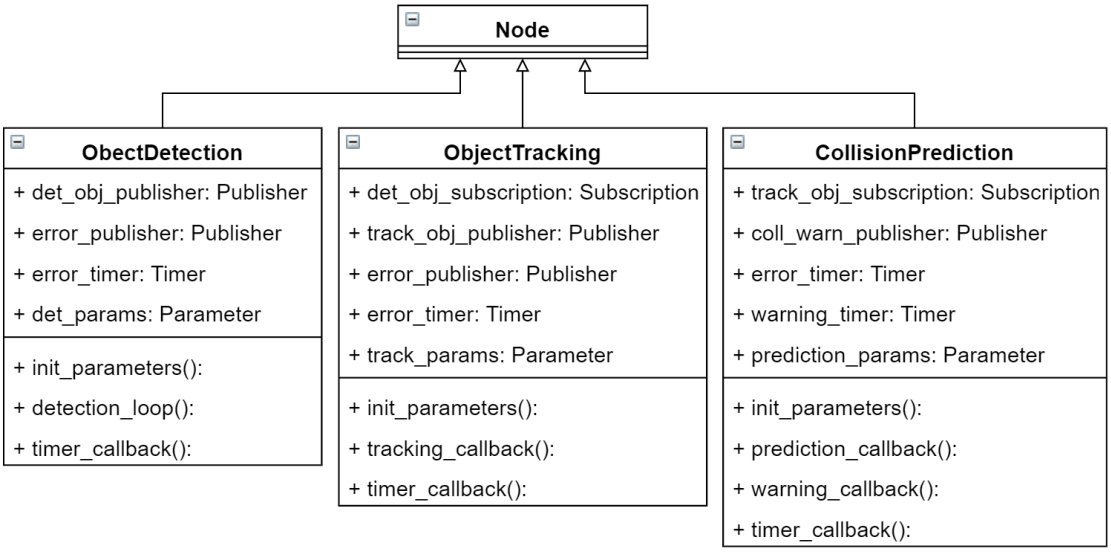


Abbildung 8.1: Klassendiagramm der Nodes

detection_loop aufgerufen, in der die Sensordatenverarbeitung und Objekterkennung zyklisch abgearbeitet wird (Listing 8.1). Die Algorithmen für die Objekterkennung und für die Sensordatenverarbeitung sind in den Klassen *ObjectDetector* und *DataProcessing* umgesetzt.

Listing 8.1: detection_loop

```
1     def detection_loop(self):
2         # continuous loop as long as rclpy is running
3         while rclpy.ok():
4             # detect objects from depth-frame
5             self.object_detector.detect_objects()
6             # reset Error Timer
7             self.error_timer.reset()
8             # send objects to topic detected_objects
9             self.generate_and_publish_objects()
```

Die zyklische Abarbeitung arbeitet nach einem festen Takt, der durch die Einstellung der FPS festgelegt wird. Die Berechnungen und das Senden werden schneller abgearbeitet, als neue Frames gelesen werden. Die erkannten Objekte werden über den *det_obj_publisher* an das Topic *detected_objects* gesendet. Als Zeitstempel für die Kollisionsvorhersage wird aus jedem Frame die Auslesezeit und für die Objektverfolgung die Framenummer mit ausgelesen. In der Klasse wird ein Timer *error_timer* implementiert, der nach jedem

Ablaufen einer vorgegebenen Zeit eine Fehlermeldung sendet. Dieser Timer wird beim Auslesen eines neuen Depth-Frames zurückgesetzt. Der Timer wird so eingestellt, dass er eine Fehlermeldung sendet, wenn drei Frames in Folge nicht ausgelesen werden. Da die Funktionen unabhängig voneinander sind, werden nach dem Senden der Fehlermeldung weiter Frames ausgelesen.

8.3 ObjectTracking

Die Logik für die Objektverfolgung ist in der Klasse *ObjectTracking* umgesetzt und wird nach Abbildung 6.5 implementiert. Anders als bei der *ObjectDetection* werden die Methoden alle eventgesteuert aufgerufen. Für die Überwachung wird der Timer *error_timer* implementiert, welcher zurückgesetzt wird, wenn eine Message von der Subscription *det_obj_subscription* eingelesen wird. In diesem Fall wird die Methode *tracking_callback* aufgerufen (Listing 8.2).

Listing 8.2: tracking_callback

```
1     def tracking_callback(self, msg):
2         self.error_timer.reset()
3         # Creates state vector from message
4         self.read_detection_message(msg)
5         # Only executed in first run to initialise old_frame_number
6         if self.old_frame_number is None:
7             self.old_frame_number = self.frame_number
8         num_of_predictions = self.frame_number - self.old_frame_number
9         if not 1 <= num_of_predictions <= MAX_DROPPED_FRAMES:
10            num_of_predictions = 1
11        # executes object tracking
12        self.sort_tracker.update(self.detections, num_of_predictions)
13        # generates tracked_message from sort_tracker and publishes it
14        self.publish_tracked_objects()
15        # set old_frame_number for next detections
16        self.old_frame_number = self.frame_number
```

Um zu verhindern, dass während des Ausführens der Objektverfolgung der *error_timer* abläuft und eine Fehlermeldung generiert wird, wird dieser als erstes zurückgesetzt. Anschließend wird die Objektverfolgung mit dem *sort_tracker* ausgeführt.
Die Variable *msg*, die an die Methode übergeben wird, enthält die Informationen, welche von der *ObjectDetection* gesendet werden. Die einzelnen Werte werden aus der Va-

riable gelesen und der Zustandsvektor *detections*, welcher zum Ausführen des SORT-Algorithmus verwendet wird, wird erzeugt. Als nächstes wird die Variable *frame_number* ausgelesen. Der Wert wird vom Wert der vorherigen Ausführung abgezogen, um zu ermitteln, wie viele Frames zwischen der aktuellen und letzten Objektverfolgung liegen. Die Differenz wird in *num_of_predictions* gespeichert. Hat die Variable *old_frame_number* den Wert None, wird die Methode zum ersten Mal ausgeführt und die Variable muss erst initialisiert werden. *num_of_predictions* wird an den SORT-Algorithmus übergeben und gibt an, wie viele Prädiktionsschritte die integrierten Kalman-Filter ausführen müssen, bevor der Update-Schritt ausgeführt wird. Dies ist zwingend notwendig, damit die Geschwindigkeiten korrekt berechnet werden. Nach dem Ausführen des SORT-Algorithmus wird die Methode *publish_tracked_objects* aufgerufen, in der die Message *TrackedObjects* generiert und mit dem *track_obj_pulisher* gesendet wird.

8.4 CollisionPrediction

Die Implementierung der Logik für die Kollisionsvorhersage erfolgt wie schon bei den anderen Nodes nach dem zugehörigen Aktivitätsdiagramm aus Abbildung 6.6 in der Klasse *CollisionPrediction*. Für die Klasse werden zwei Subscriptions erzeugt, *error_subscription* und *objects_subscription*, welche zum einen die Fehlermeldungen und zum anderen die verfolgten Objekte beinhalten. Der Publisher *warning_publisher* sendet die aktuelle Warnmeldung an das Topic *warning_topic*.
Die Methode *prediction_callback* wird jedes mal aufgerufen, wenn eine Nachricht von *objects_subscription* empfangen wird oder wenn der Timer *prediction_timer* abgelaufen ist (Listing 8.3).

Listing 8.3: prediction_callback

```
1     def prediction_callback(self, msg=None):
2         # msg is None, if method is called from Timer
3         if msg is not None:
4             # reset the error and prediction timer
5             self.error_timer.reset()
6             self.prediction_timer.reset()
7             # update positions and velocities from tracked objects
8             self.read_tracking_msg(msg)
9         # if self.detections is not initialised, collision cannot be
    predicted
10        if self.detections is not None:
```

```
11              # execute prediction
12              self.collision_warning = self.collision_predictor.
    predict_collision(
13                  self.detections,
14                  self.tracking_time
15              )
16          else:
17              self.collision_warning = Danger.Danger
18              # publish calculated collision warning
19          self.publish_collision_warning()
```

Wird die Methode von dem Timer aufgerufen, wird der Zustandsvektor nicht aktualisiert, bevor die Kollisionsvorhersage ausgeführt wird. Wird die Methode von der *object_ subscription* aufgerufen, werden der *prediction_ timer* und der *error_ timer* zurückgesetzt, anschließend werden die Zustände aktualisiert und die Kollisionsvorhersage ausgeführt.

9 Auswertung

In diesem Kapitel werden die Ergebnisse der Arbeit vorgestellt und anschließend anhand der Anforderungen bewertet.

9.1 Ergebnisse

Die Ergebnisse der Kollisionsvermeidung werden in separaten Abschnitten betrachtet. Zuerst wird die Ausführungszeit der einzelnen Algorithmen und die Ausführungszeit des Gesamtsystems vorgestellt. Anschließend wird die Genauigkeit der einzelnen Komponenten dargestellt. Um die Funktion der Algorithmen zu überprüfen, werden verschiedene Szenarien mit dem Teststand und dem Roboter aufgenommen und anschließend ausgewertet. Die Szenarien beinhalten verschiedene Objekte, die sich in dem FOV des Roboters befinden oder bewegen, sich aus dem FOV bewegen oder in das FOV bewegen.

9.1.1 Ausführungszeiten

In diesem Teilabschnitt werden die Ausführungszeiten der Algorithmen auf einem Windows PC und einem Raspberry Pi 4 gegenübergestellt. Die Ausführungszeiten sind in Abbildung 9.1 dargestellt. Die Ausführungszeiten der Module auf dem Windows PC sind geringer als die auf dem Raspberry Pi. Durch die Architektur des Programm können die einzelnen Module parallel abgearbeitet werden. Die Objekterkennung kann nach dem Senden der Objektlisten den nächsten Datenstrom auslesen und verarbeiten. Dadurch kann eine Abtastfrequenz von 15Hz erreicht werden, auch wenn die Gesamtausführungszeit 72ms beträgt. Mit dem PC ist sogar eine Abtastung von 60Hz möglich.

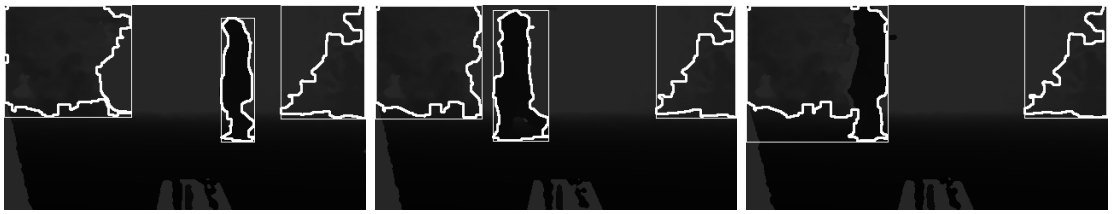
Tabelle 9.1: Ausführungszeiten der Kollisionsvermeidung

Zeitmessung	PC / Numba	Rapsberry Pi / Numba
Objekterkennungsalgorithmus	27ms / 14ms	120ms / 43ms
Objektverfolgung	7ms	9ms
Kollisionsvorhersage	3ms	5ms
Austausch der Nachrichten	15ms	7ms
Gesamtausführungszeit	67 ms / 54ms	137ms / 72ms
Erreichbare Abtastung	30Hz / 60Hz	5Hz / 15Hz

9.1.2 Objekterkennung

Um die Genauigkeit der Objekterkennung bewerten zu können, werden die Ergebnisse der Objekterkennung in vier verschiedene Kategorien *True Positive* (TP), *True Negative* (TN), *False Positive* (FP) und *False Negative* (FN) eingeteilt. *Positive* gibt an, dass die Objekterkennung einen Bereich in dem Bild als Objekt erkennt. *Negative* gibt an, dass die Objekterkennung einen Bereich als Hintergrund erkennt. *True* und *False* geben an, ob die Einstufung richtig ist oder falsch. Befindet sich beispielsweise eine Person im Bild und sie wird als Objekt erkannt, ist sie als TP einzuordnen. Wird die Person nicht erkannt, ist sie als FN einzustufen. Wird der Hintergrund als Hintergrund erkannt, ist er als TN einzustufen und, wenn er als Objekt erkannt wird, als FP.
Um eine Objekterkennung zu testen, können Datenbanken zum Testen der Algorithmen, wie die Datenbank in [12], verwendet werden. In diesen Datenbanken sind Testdaten hinterlegt, in denen Objekte in Kategorien eingestuft sind. Umfangreiche Testdaten zu erzeugen ist sehr aufwendig. Ein großer Fokus liegt auf der Umgebung Straßenverkehr und Personenverkehr in urbanen Szenen, weshalb diese für eine Validierung dieser Arbeit nicht sehr gut geeignet sind.
In dieser Arbeit werden zum Validieren der Objekterkennung verschiedene Testszenarien aufgenommen und anschließend ausgewertet. In den Testszenarien werden verschiedene Objekte aufgenommen und ausgewertet, wie gut diese Objekte erkannt werden. Bei der Auswahl der Objekte wird darauf geachtet, dass Objekte verwendet werden, die verschiedene Eigenschaften aufweisen und in einem Park vorkommen. Eigenschaften, die bei den Ergebnissen berücksichtigt werden, sind die Größe, die Beschaffenheit der Oberfläche und die geometrische Form. Neben der Berücksichtigung verschiedener Objekteigenschaften wird untersucht, wie gut Objekte erkannt werden, die komplett oder teilweise verdeckt werden. Da es nicht möglich ist, in dem Rahmen dieser Arbeit alle Szenarien zu überprüfen, werden exemplarisch Objekte analysiert, die eine Gruppe von Eigenschaften vereinen.

Referenzobjekte sind Menschen, ein Fahrrad, ein Fahrradfahrer, eine Bank, ein Baum, eine Hecke und ein Ball. Als Untergrund wird ein gepflasterter Weg, der sehr ähnliche Eigenschaften aufweist wie ein Schotterweg, und eine Rasenfläsche untersucht. Die Erkennung der Objekte wird mit Videosequenzen durchgeführt, in denen sich die Objekte durch die Szene bewegen. Die Objekte werden sehr gut erkannt, wodurch es nur in seltenen Fällen zu einer FN Aussage kommt. Mit FN wird der Ball eingestuft, wenn er sich weiter als $z_r \approx 2$ m von dem Roboter entfernt und sich in hohem Gras befindet. Das liegt daran, dass der Ball im Verhältnis zu den anderen Objekten sehr klein ist und durch seine runde Oberfläche keine konstanten Entfernungswerte liefert. Auf gepflastertem Boden wird der Ball bis zu einer Entfernung von $z_r \approx 4m$ sicher erkannt.
Werden Objekte teilweise oder ganz von anderen Objekten verdeckt, können die Objekte nicht klar voneinander getrennt werden. In Abbildung 9.1 ist eine Person dargestellt, die sich von rechts nach links durch die Szene bewegt. Neben der Person sind in der Abbildung zwei Hecken zu erkennen, die ebenfalls als Objekte erkannt werden. Die Bilder in der Abbildung sind nach dem zeitlichen Verlauf von links nach rechts angeordnet. In Abbildung 9.1a befindet sich die Person, die sich durch die Szene bewegt, mittig zwischen den Objekten und es werden alle Objekte getrennt voneinander erkannt. In Abbildung 9.1b befindet sich die Person sehr nah an der linken Hecke, die Objekterkennung kann die Objekte jedoch noch voneinander trennen. In Abbildung 9.1c verdeckt die Person einen Teil der Hecke und der Algorithmus kann nicht mehr zwischen der Person und der Hecke unterscheiden. Die beiden Objekte werden als ein Objekt erkannt.



(a) Person mittig (b) Person bei linker Hecke (c) Person verdeckt Hecke

Abbildung 9.1: Teil eines Objektes verdeckt

9.1.3 Objektverfolgung

Die Genauigkeit der Objektverfolgung wird für die beiden in Abschnitt 7.3 beschriebenen Fälle vorgenommen. Wie in [2] beschrieben, hängt die Genauigkeit der Objektverfolgung

direkt mit der Genauigkeit der Objekterkennung zusammen. Die Ergebnisse der Objektverfolgung werden anhand der Objekte durchgeführt, die von der Objekterkennung bereitgestellt werden. Zuerst werden die Ergebnisse der beiden Verfahren gegenübergestellt und anschließend die um die z-Koordinate erweiterte Methode genauer betrachtet.

Vergleich Konzept und anschließende Erweiterung

Der direkte Vergleich der Methoden wird mit und ohne Berücksichtigung der z-Koordinate anhand einer, in Abbildung 9.2 und Abbildung 9.3 dargestellten, Bildsequenz vorgenommen. Der erste Teil der Bildsequenz, welcher in Abbildung 9.2 dargestellt ist, zeigt zwei

(a) Ein Objekt im Bild (b) Zweites Objekt am Rand (c) Zwei Objekte im Bild

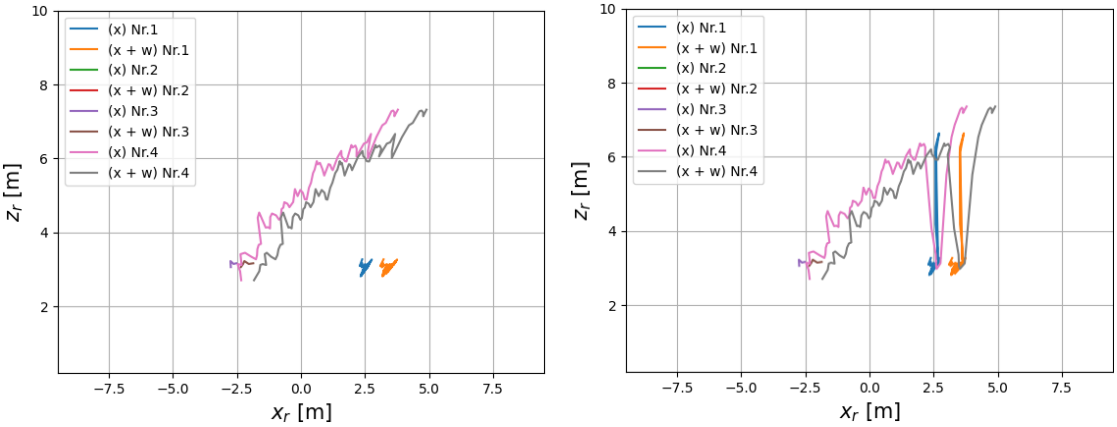
Abbildung 9.2: Bildsequenz mit zwei Personen im Bild

Objekte. Ein Objekt, eine Person, steht fest im rechten Bereich des Bildes und das andere Objekt, ebenfalls eine Person, bewegt sich von links in das Bild. Ausschnitte des zweiten Teils der Bildsequenz sind in Abbildung 9.3 dargestellt. Die erste Person, im rechten Bereich des Bildes, steht nach wie vor fest an der gleichen Position. Die zweite Person bewegt sich von links nach rechts durch das Bild.
Die Ergebnisse der Objektverfolgung, angewendet auf diese Bildsequenz, sind in Abbildung 9.4 dargestellt. Abbildung 9.4a zeigt die Ergebnisse unter Berücksichtigung der z-Koordinate und Abbildung 9.4b zeigt die Ergebnisse ohne Berücksichtigung der z-

(a) Objekt links (b) Objekt mittig (c) Objekt rechts

Abbildung 9.3: Bildsequenz mit einer stehenden und einer bewegten Person

Koordinate. Die Bounding Boxes werden in der x, z-Ebene des Roboterkoordinatensys-



(a) Erweiterung der IOU um z-Achse

(b) Verfolgung ohne z-Achse

Abbildung 9.4: Position von verfolgten Objekten im Roboterkoordinatensystem

tems dargestellt. Für die Darstellung der Objekte werden jeweils die Positionen $P(x, z)$ und $P(x + w, z)$ abgebildet, um die Breite der Objekte mit darzustellen. In der Legende ist zu erkennen, dass der Objektvervolgungsalgorithmus vier verschiedene Objekte verfolgt. Das erste Objekt ist die rechte Personen im Bild. Der Mittelpunkt der Person befindet sich auf dem Punkt $P(x, z) = (2,5\ \text{m}, 3\ \text{m})$ und bewegt sich nicht. Sie befindet sich während der gesamten Bildsequenz im FOV der Kamera. Die drei weiteren verfolgten Objekte stellen alle die zweite Person dar, die sich von links in das FOV der Kamera bewegt. Diese fehlerhafte Zuordnung entsteht zu Beginn der Bildsequenz. In Abbildung 9.2a befindet sich nur das erste Objekt im Bild. In Abbildung 9.2b ist am linken Rand zu erkennen, dass sich das zweite Objekt ins Bild bewegt. Das Objekt ist aber nur zu einem kleinen Teil im Bild, weshalb die Objektverfolgung das Objekt in dem darauffolgenden Bild, Abbildung 9.2c, nicht wieder erkennt und ein neues Objekt erzeugt wird. Erst wenn das Objekt komplett im Bild ist, in Abbildung 9.3a, kann es über die gesamte Sequenz verfolgt werden. Für den Abschnitt, in dem das Objekt komplett im Bild ist, wird dem zweiten Objekt die Objektnummer Nr. 4 zugeordnet.

Für den Zeitraum, in dem sich beide Objekte komplett in der FOV befinden, kann der erweiterte SORT-Algorithmus unter der Verwendung der z-Information, die Objekte in jedem Bild verfolgen. Der Algorithmus ohne die z-Informationen kann die Objekte hingegen nicht mehr korrekt zuordnen, in dem Moment wo sich das bewegende Objekt auf die gleiche x-Position bewegt. In Abbildung 9.3c hat sich das Objekt so weit entlang der x-Achse bewegt, dass der Algorithmus die Objekte wieder richtig zuordnen kann.

Genauigkeit der Objektverfolgung

Um die Genauigkeit der Objektverfolgung zu überprüfen, werden verschiedene Szenarien betrachtet. Die Szenarien beinhalten starre Objekte und einen starren Roboter, bewegte Objekte und einen starren Roboter, starre Objekte und einen bewegten Roboter und bewegte Objekte und einen bewegten Roboter. Der Algorithmus benötigt jedoch ca. 4-5 Zyklen, bis sich das Kalman-Filter des jeweiligen Objektes auf die Geschwindigkeit der Objekte angepasst hat. In diesen Zyklen schwanken die Geschwindigkeitswerte des verfolgten Objektes sehr stark. Dieser Effekt ist bei jedem getesteten Szenario zu erkennen. Bei einem starren Roboter und einem starren Objekt kann die Position und die Geschwindigkeit sehr gut verfolgt werden. Die Abweichungen sind minimal und können vernachlässigt werden. In Abbildung 9.5 ist die Bewegung einer Person in einer Entfernung von $z \approx 2,5$ m dargestellt. Die Abbildung zeigt zum einen den Positionsverlauf in

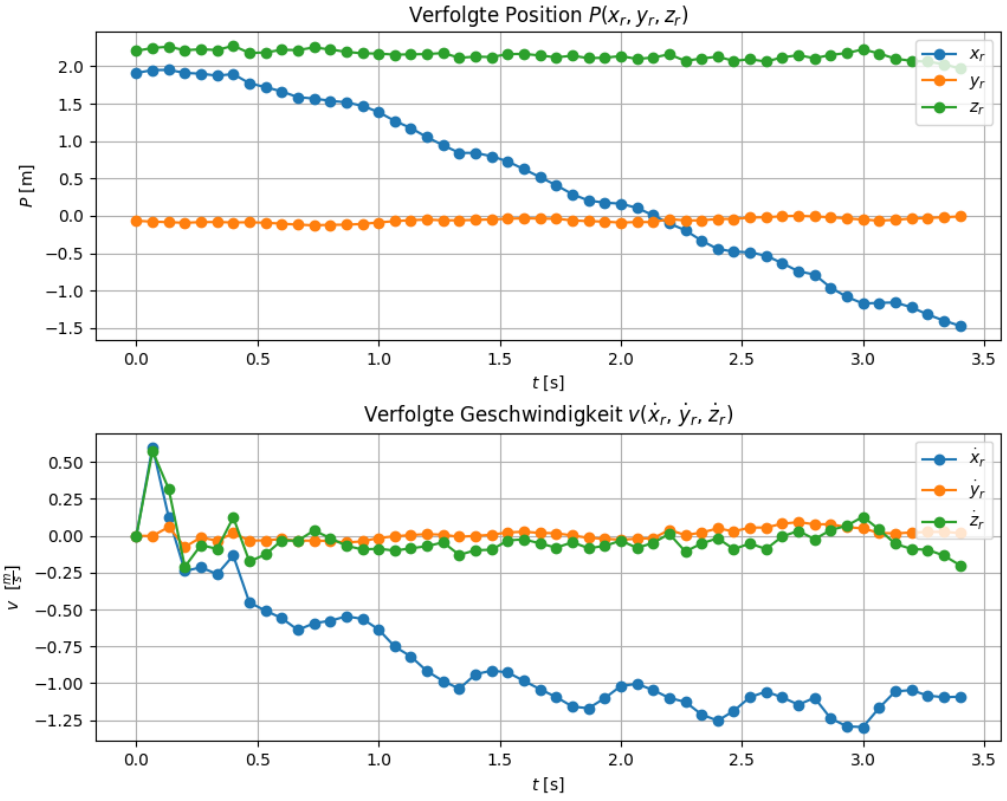


Abbildung 9.5: Bewegung einer Person entlang der x_r-Koordinate

Roboterkoordinaten und zum anderen die ermittelte Geschwindigkeit, welche über das Kalman-Filter berechnet wird. Die Position und die Geschwindigkeit wird anhand der

Mittelpunktes der Bounding Box berechnet. Aus dem Positionsverlauf der x_r-Koordinate ist zu entnehmen, dass sich die Person von rechts nach links durch das Bild bewegt. Beim Gehen einer Person verändert sich die Form der Bounding Box, da die Beine und Arme sich vor und zurück bewegen. Dieser Effekt ist in Abbildung 9.1a und in Abbildung 9.1b sehr gut zu erkennen. Durch die Veränderung der Form der Bounding Box verschiebt sich der Mittelpunkt, wodurch die unkonstante Geschwindigkeit entlang der x_r-Koordinate entsteht.

9.1.4 Kollisionsvorhersage

Neben der Zykluszeit wird die Kollisionsvorhersage auf die Funktionalität überprüft. Für die Überprüfung der Kollisionsvorhersage werden verschiedene Szenarien erstellt, die mit Unittests überprüft werden. Anschließend wir die Kollisionsvorhersage im realen Betrieb getestet. In Abbildung 9.6 ist der Verlauf einer Kollisionsvorhersage dargestellt. In der Abbildung ist die Bewegung des Objektes in x, z-Koordinaten dargestellt. Im

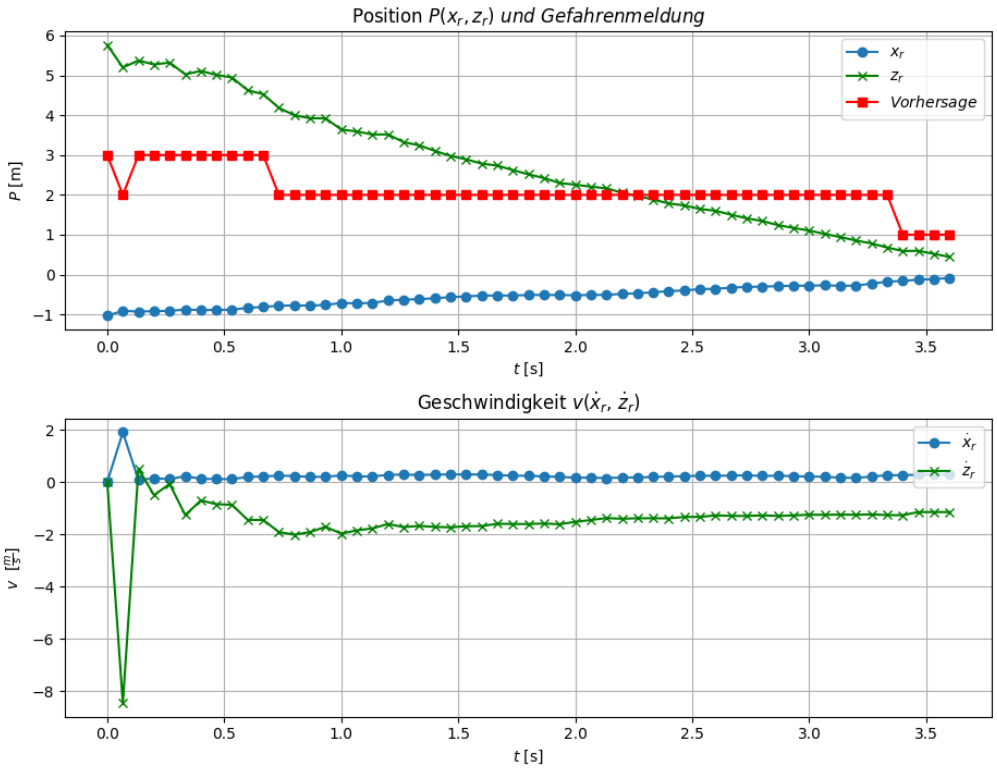


Abbildung 9.6: Objekt, welches eine Kollision verursacht

oberen Plot ist für jeden Zeitschritt zusätzlich die Kollisionswarnung mit abgebildet. Um die Kollisionswarnung im Plot darzustellen, wird die Meldung *Keine Gefahr* mit drei, *Potentielle Gefahr* mit zwei und *Gefahr* mit 1 hinterlegt. In dieser Abbildung ist sehr gut zu erkennen, dass die Objektverfolgung zu Beginn sehr stark von den Messwerten abhängt und die Geschwindigkeiten sehr stark schwanken. Obwohl das Objekt über 5 m entfernt ist, wird die Meldung *Potentielle Gefahr* generiert, da das Objekt mit der ermittelten Geschwindigkeit innerhalb des zeitlichen Horizonts in die ROI bewegt. Nach dem Einschwingen wird konstant die Meldung *Keine Gefahr* ausgegeben, da sich das Objekt auf den Roboter zu bewegt, aber sich nicht innerhalb des zeitlichen Horizonts in die ROI bewegt. Ist das Objekt in dem Bereich, dass es die ROI innerhalb von *dt* erreicht, wird die Meldung *Potentielle Gefahr* generiert. Dies tritt bei $t \approx 0,6$ s ein. Für circa 2,6 s wird kontinuierlich die Warnung *Potentielle Gefahr* generiert. Da die Geschwindigkeit des Objektes abnimmt, bis es sich in die ROI bewegt, entspricht dieser Wert nicht genau *dt*. Nach $t \approx 3,4$ s befindet sich das Objekt in der ROI und es wird die Meldung *Gefahr* generiert.

9.2 Diskussion

In der Diskussion werden die Ergebnisse mit den Anforderungen verglichen und die Ergebnisse bewertet. Die Anforderungen werden mit *Erfüllt* (✓), *Teilweise Erfüllt* (○) und *Nicht Erfüllt* (×) eingestuft.
Im ersten Schritt wird die Bewertung der funktionalen Anforderungen in Tabelle 9.2 vorgenommen. Die Funktionalen Anforderungen an die Kollisionsvermeidung sind erfüllt. Alle Funktionalitäten, die in den Anforderungen gestellt wurden sind umgesetzt.

Tabelle 9.2: Bewertung der funktionalen Anforderungen

ID	Kommentar	Erfüllt
1	Objekterkennung im Modul *Object Detection* umgesetzt	✓
2	Kollisionsvorhersage im Modul *Collision Prediction* umgesetzt	✓
3	Objektverfolgung im Modul *Object tracking* umgesetzt	✓
4	Kollisionsmeldung wird an das Topic *Collision Warning* gesendet	✓

Die Spezifikationen der funktionalen Anforderungen werden in Tabelle 9.3 bewertet. Messungen haben gezeigt, dass es möglich ist, Objekte, die sich schnell bewegen, zu verfolgen und eine Kollisionsvorhersage durchzuführen. Die Ergebnisse werden innerhalb von 72 ms bereitgestellt, was es der Steuerung ermöglicht, frühzeitig eine Gefahrenmeldung an

die Steuerung zu senden. Mit dem verwendeten Algorithmus zur Objekterkennung ist

Tabelle 9.3: Bewertung der Anforderungen an das System

ID	Kommentar	Erfüllt
5	Verfolgung von Objekten mit $v = z_{max}$ ist mit der Abtastung möglich	✓
6	Sind Teile von Objekten von anderen Objekten verdeckt, werden die Objekte als eins erkannt.	○
7	Die Kamera ist nach Herstellerangaben für den Outdoor Einsatz geeignet	✓
8	Mit der Auflösung 424×240 ist die Kamera für Entfernungen von 0,2m-10m geeignet	✓
9	Mit dem Öffnungswinkel und der Positionierung der Kamera, kann die ROI in Fahrtrichtung des Roboters aufgenommen werden	✓
10	An den Grenzen des vorgegebenen Arbeitsbereich kann der Algorithmus Objekten nicht komplett zuverlässig erkennen.	○

es möglich, mehrere Objekte in einer aufgenommenen Szene zu erkennen, es ist jedoch nicht möglich, Objekte die sich teilweise verdecken, voneinander zu trennen. Die Kamera ist für Outdoor Anwendungen geeignet und hat mit der passenden Parametrisierung einen Arbeitsbereich von 0,2m-10m. Befinden sich Objekte mit unebenen Oberflächen wie Personen im Bereich $z_r = 0,2$ m, kann der Objekterkennungsalgorithmus das Objekt nicht mehr als ein einzelnes erkennen und erkennt mehrere kleine Objekte. Da die Positionsangaben der kleinen Objekte jedoch weiterhin korrekt ausgegeben werden, kann die Kollision trotzdem korrekt vorhergesagt werden.
Die zusätzlichen Anforderungen, die sich durch die Stakeholderanalyse ergeben, werden in Tabelle 9.4 bewertet.

Tabelle 9.4: Zusätzliche Anforderungen der Stakeholder

ID	Kommentar	Erfüllt
11	DSGVO wird eingehalten	✓
12	Die Software ist in separate ROS2 Packages aufgeteilt	✓
13	Die Module werden über ROS2 Topics miteinander verbunden	✓
14	ROS2 kann auf Windows, Linux und OSX installiert werden	✓
15	Die Neuanschaffungen haben das Budget nicht überschritten	✓
16	Die erlangten Erkenntnisse sind in Form dieser Arbeit und des Quellcodes dokumentiert	✓
17	Die Hardware nicht verändert und kann wiederverwendet werden	✓

Die DSGVO wird nach [14] eingehalten, da die Personen in dem Bild nur durch die Stereoinformationen repräsentiert werden und daraus keine Personenbezogenen Daten ermittelt werden können. Durch die Verwendung von ROS2 ist die Software so modular aufgebaut, dass die einzelnen Module separat voneinander ausgeführt, exportiert oder ausgetauscht werden können, sofern die Schnittstellen von anderen Services bedient werden. Durch die Verwendung von ROS2 besteht die Möglichkeit, die Software auf den gängigen Betriebssystemen zu installieren und auszuführen. Durch die Verwendung von Materialien, die von Herrn Hensel bereitgestellt wurden und nach der Arbeit für andere Projekte verwendet werden können, wurde das Budget nicht überschritten.

10 Fazit und Ausblick

In dieser Arbeit wurde ein System entwickelt, mit dem es möglich ist, eine Kollision zwischen verschiedenen Objekten und einem Roboter unter der Verwendung von 3D-Bilddaten und effizienten Algorithmen frühzeitig zu erkennen. Das System wurde für den Outdoor Einsatz konzipiert und liefert für weitläufige freie Flächen sehr gute Ergebnisse. Kommen viele Objekte in der Szene vor oder treten große Objekte im Hintergrund auf, können die erkannten Objekte nicht mehr auseinander gehalten werden und die Positions- und Bewegunsschätzung liefert falsche Werte. Für den realen Einsatz ist hier mit weiteren Messungen zu überprüfen, welchen Einfluss diese Aspekte auf eine sichere Fahrt des Roboters haben. Kommen die Objekte, die andere verdecken, in den Bereich der ROI, verdecken diese in den meisten Fällen Objekte im Hintergrund komplett, wodurch eine Falschberechnung der Positionen nicht mehr auftritt.
Die Verwendung des SORT-Algorithmus und der Erweiterung auf den 3D-Raum ermöglicht es, die Positionen und Geschwindigkeiten der Objekte bezogen auf den Roboter genau genug zu schätzen, um Kollisionen frühzeitig zu erkennen.
Die Verwendung von Python als Programmiersprache hat sich zwischenzeitlich trotz der Verwendung der Bibliothek NumPy als problematisch dargestellt, da es nicht möglich war alle Berechnungen effizient umzusetzen und alle Module in der gewünschten Zeit auszuführen. Erst mit der Verwendung der Numba-Bibliothek ist es gelungen, die Ausführungszeiten der Algorithmen so zu verbessern, dass die gewünschten Zykluszeiten erreicht werden konnten. Bei der Verwendung der Numba-Bibliothek muss jedoch darauf geachtet werden, welche Funktionen beschleunigt werden und welche nicht, da es nicht nöglich ist alle funktionen mit Numba zu beschleunigen.
Die Umsetzung des Projektes mit ROS2 bringt einen relativ hohen Initialaufwand mit sich. Ist das System korrekt aufgesetzt und die benötigten Packages sind korrekt installiert, ist die Verwendung von ROS2 sehr angenehm. Auf dem Windows PC ist das erneute Bauen von Packages problemlos möglich, auf dem Raspberry Pi hat es sich teilweise als schwierig herausgestellt. Das erneute Bauen war teilweise nur möglich, wenn die Dateien aus dem *Build*- und *Install*-Pfad gelöscht wurden. Für große Projekte ist ROS2 sehr

sinnvoll, da vor allem die Kommunikation zwischen den Modulen sehr komfortabel umzusetzen ist, bei kleinen Projekten muss man jedoch abschätzen, ob sich der Initialaufwand lohnt.
Um den einen Einsatz des Systems in dem realen Betrieb zu ermöglichen, muss für die Objekterkennung eine weiterführende Validierung durchgeführt werden. Die Ergebnisse resultieren aus konstruierten Szenarien, die eine vollumfängliche Bewertung nicht ermöglichen. Um mehr Sicherheit für die Funktion zu erlangen sind mehr Messdaten notwendig.
Um die Objekterkennung zu verbessern, wäre es möglich, eine weitere Funktion einzubauen, die teilweise verdeckte Objekte voneinander trennt. Dafür könnte eine Clusteranalyse durchgeführt werden, in der die Häufung von Tiefenwerten analysiert wird. Eine andere Möglichkeit besteht darin, die Tiefenwerte der Objekte in einem Histogramm darzustellen und dort Objekte anhand eines Schwellwerts zu segmentieren.
Mit dem Einsatz eines Roboters, der sich so schnell bewegt wie der ExoMy, könnte das System in Innenräumen eingesetzt werden, wenn der Arbeitsbereich der Kamera angepasst wird.

Literaturverzeichnis

[1] Aguilar, Wilbert G. ; Casaliglla, Verónica P ; Polit, Jose L.: Obstacle avoidance based-visual navigation for micro aerial vehicles. In: *Electronics* 6 (2017), Nr. 1, S. 10

[2] Bewley, Alex ; Ge, Zongyuan ; Ott, Lionel ; Ramos, Fabio ; Upcroft, Ben: Simple online and realtime tracking. In: *2016 IEEE International Conference on Image Processing (ICIP)* (2016), Sep. – URL http://dx.doi.org/10.1109/ICIP.2016.7533003

[3] Beyerer, Jürgen ; León, Fernando P. ; Frese, Christian: *Automatische Sichtprüfung: Grundlagen, Methoden und Praxis der Bildgewinnung und Bildauswertung.* Springer-Verlag, 2016

[4] Bibliographisches Institut: *Duden, Bedeutung Park.* 2021. – URL https://www.duden.de/rechtschreibung/Park. – Zugriffsdatum: 2021-12-10

[5] Bosch, Siegfried: Lineare Abbildungen. In: *Lineare Algebra.* Springer, 2014, S. 51–82

[6] Burger, Wilhelm ; Burge, Mark J.: *Digitale Bildverarbeitung: Eine Algorithmische Einführung Mit Java.* Springer-Verlag, 2015

[7] Eich, Walter ; Bachmann, Matthias: *Advanced Software Engineering1, Requirements Engeneering.* Züricher Fachhoschschule, 2019

[8] Einfach Technik Daimler AG: *Einfach Technik: Hochautomatisiertes Fahren mit dem DRIVE PILOT.* 2021. – URL https://www.daimler.com/magazin/technologie-innovation/einfach-technik-drive-pilot.html. – Zugriffsdatum: 2021-11-01

[9] Erhardt, Max: *GPL-3.0 License.* 2007. – URL https://github.com/esa-prl/ExoMy_Software/blob/master/LICENSE. – Zugriffsdatum: 2021-11-17

[10] European Space Agency: *3D print your own Mars rover with ExoMy*. 2020. – URL https://www.esa.int/Enabling_Support/Space_Engineering_Technology/3D_print_your_own_Mars_rover_with_ExoMy. – Zugriffsdatum: 2021-11-17

[11] Geiger, Andreas: *Kitti-Multi Object Tracking*. 2021. – URL http://www.cvlibs.net/datasets/kitti/eval_tracking_detail.php?result=4b2ae9426f6584eff389ba455dec2069d9d17ca3. – Zugriffsdatum: 2021-10-31

[12] Geiger, Andreas ; Lenz, Philip ; Stiller, Christoph ; Urtasun, Raquel: *The KITTI Vision Benchmark Suite: Object Tracking Evaluation*. 2021. – URL http://www.cvlibs.net/datasets/kitti/eval_tracking.php. – Zugriffsdatum: 2021-12-13

[13] Gonzalez-Garcia, Alejandro ; Collado, Ivana ; Cuan-Urquizo, Rodolfo ; Reyes, Roberto ; Garrido, Leonardo: A 3D Vision Based Obstacle Avoidance Methodology for Unmanned Surface Vehicles. In: *XXI Congreso Mexicano de Robótica* Bd. 2019, 2019

[14] Hartwig, Matthias ; LL.M., Benedicte M. ; Schumacher, Oskar: *Rechtliche Rahmenbedingungen für den Einsatz von autonomen Robotern in Assistenzfunktionen*. IKEM, 2020

[15] Hertzberg, Joachim ; Lingemann, Kai ; Nüchter, Andreas: *Mobile Roboter: Eine Einführung aus Sicht der Informatik*. Springer-Verlag, 2012

[16] Intel Corporation: *Realsense Cameras*. 2020. – URL https://www.intelrealsense.com/compare-depth-cameras/. – Zugriffsdatum: 2021-09-14

[17] Intel Corporation: *Realsense Cameras datasheets*. 2020. – URL https://www.intelrealsense.com/wp-content/uploads/2020/06/Intel-RealSense-D400-Series-Datasheet-June-2020.pdf. – Zugriffsdatum: 2021-09-14

[18] Jeyachandran, Satish: *Introducing the 5th-generation Waymo Driver: Informed by experience, designed for scale, engineered to tackle more environments* . 2020. – URL https://blog.waymo.com/2020/03/introducing-5th-generation-waymo-driver.html. – Zugriffsdatum: 2021-11-01

[19] Kalman, Rudolph E. ; Bucy, Richard S.: New results in linear filtering and prediction theory. (1961)

[20] Lin, Jiahao ; Zhu, Hai ; Alonso-Mora, Javier: Robust vision-based obstacle avoidance for micro aerial vehicles in dynamic environments. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)* IEEE (Veranst.), 2020, S. 2682–2688

[21] Marchthaler, Reiner ; Dingler, Sebastian: Kalman-Filter. In: *Einfuhrung in die Zustandsschätzung und ihre Anwendung fur eingebettete Systeme. Berlin* (2017)

[22] Maurer, Markus ; Gerdes, J C. ; Lenz, Barbara ; Winner, Hermann: *Autonomes Fahren: technische, rechtliche und gesellschaftliche Aspekte*. Springer Nature, 2015

[23] Oleynikova, Helen ; Honegger, Dominik ; Pollefeys, Marc: Reactive avoidance using embedded stereo vision for MAV flight. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)* IEEE (Veranst.), 2015, S. 50–56

[24] Open Robotics: *ROS2 Documentation*. 2021. – URL https://docs.ros.org/en/foxy/Releases.html. – Zugriffsdatum: 2021-10-11

[25] OpenCV: *Open Source Computer Vision Function Documentation Canny()*. – URL https://docs.opencv.org/3.4/dd/d1a/group__imgproc__feature.html#ga2a671611e104c093843d7b7fc46d24af. – Zugriffsdatum: 2022-01-09

[26] SAE International: *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*. 2021

[27] Song, Wenjie ; Yang, Yi ; Fu, Mengyin ; Qiu, Fan ; Wang, Meiling: Real-Time Obstacles Detection and Status Classification for Collision Warning in a Vehicle Active Safety System. In: *IEEE Transactions on Intelligent Transportation Systems* 19 (2018), Nr. 3, S. 758–773

[28] Szeliski, Richard: *Computer Vision: Algorithms and Applications*. Bd. 1st Edition. Springer Verlag, 2011

[29] Yang, Cong ; Jun-Jian, Peng ; Jing, Sun ; Lin-Lin, Zhu ; Yan-Dong, Tang: V-disparity based UGV obstacle detection in rough outdoor terrain. In: *Acta Automatica Sinica* 36 (2010), Nr. 5, S. 667–673

A Anhang

Anhang A1: Masterthesis als PDF-Datei
Anhang A2: ROS2 Packages mit dem Quellcode des Programms
Anhang A3: Referenzvideos

Die Anhänge sind in elektronischer Form auf einer CD abgelegt und können bei Prof. Dr. -Ing. Marc Hensel oder Prof. Dr. -Ing. Dipl. -Kfm. Jörg Dahlkemper eingesehen werden.

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. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original