



**BOSCH**

Technik fürs Leben



# Entwurf, Entwicklung und Validierung eines Light Distance and Ranging (LIDAR) Systems

## Studienarbeit

des Studiengangs Elektrotechnik, Elektronik  
an der Dualen Hochschule Baden-Württemberg Stuttgart

von

**Alexander Kehrer & Marcel Wagner**

14.06.2019

**Bearbeitungszeitraum**  
**Matrikelnummer, Kurs**  
**Ausbildungsfirma**  
**Betreuer**

01.10.2018-14.06.2019  
5813992 & 6925112, TEL16GR2  
Robert Bosch GmbH, Stuttgart  
Klaus Gosger

## Selbstständigkeitserklärung

Wir versichern hiermit, dass wir unsere Studienarbeit mit dem Thema: *Entwurf, Entwicklung und Validierung eines LIDAR Systems* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben. Wir versichern zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Stuttgart, 14.06.2019

---

Alexander Kehrer & Marcel Wagner

## **Abstract**

*TODO: deutscher Abstract....*

## Abstract

*TODO: english abstract....*

# Inhaltsverzeichnis

<b>Abkürzungsverzeichnis</b>	<b>VII</b>
<b>Abbildungsverzeichnis</b>	<b>VIII</b>
<b>Tabellenverzeichnis</b>	<b>X</b>
<b>Formelverzeichnis</b>	<b>XI</b>
<b>Listings</b>	<b>XII</b>
<b>1 Einleitung</b>	<b>1</b>
<b>2 Grundlagen Elektronik</b>	<b>3</b>
2.1 Photodioden . . . . .	3
2.1.1 Avalanche Photo Diode (APD) . . . . .	4
2.1.2 Lateral auflösende Photodiode . . . . .	5
2.2 Schrittmotoren . . . . .	6
<b>3 Grundlagen Laserentfernungsmessung</b>	<b>7</b>
3.1 Lichtlaufzeitmessung . . . . .	7
3.1.1 Grundprinzip . . . . .	7
3.1.2 Herausforderungen . . . . .	8
3.2 Phasenverschiebung / Phasenmodulation . . . . .	9
3.2.1 Grundprinzip . . . . .	10
3.2.2 Herausforderungen . . . . .	12
3.3 Triangulation . . . . .	13
3.3.1 Grundprinzip . . . . .	13
3.3.2 Herausforderungen . . . . .	15
<b>4 Stand der Technik</b>	<b>16</b>
<b>5 Machbarkeitsstudie</b>	<b>17</b>
5.1 Versuch 1: Test handelsüblicher Photodioden . . . . .	17
5.1.1 Versuchsaufbau . . . . .	17
5.1.2 Beobachtung . . . . .	19
5.1.3 Erkenntnis . . . . .	19

5.2	Versuch 2: Weiterer Test handelsüblicher Photodioden . . . . .	20
5.2.1	Versuchsaufbau . . . . .	20
5.2.2	Beobachtung . . . . .	21
5.2.3	Erkenntnis . . . . .	22
5.3	Weiteres Vorgehen . . . . .	22
5.3.1	Recherche . . . . .	22
<b>6</b>	<b>Matlab Modell</b>	<b>23</b>
<b>7</b>	<b>Mechanik</b>	<b>24</b>
7.1	Anforderungen . . . . .	24
7.2	Entwurf . . . . .	24
7.2.1	Oberer Aufbau . . . . .	24
7.2.2	Basis . . . . .	26
7.2.3	Rahmen . . . . .	27
7.2.4	Materialliste . . . . .	28
7.3	Umsetzung . . . . .	28
<b>8</b>	<b>Hardware</b>	<b>30</b>
8.1	Funktionseinheit Distanzbestimmung . . . . .	30
8.1.1	TF Mini LIDAR . . . . .	30
8.1.2	VL53L1X . . . . .	31
8.1.3	Tabellarischer Vergleich der Sensoren . . . . .	33
8.2	Funktionseinheit Ausrichtung des Sensors . . . . .	34
8.2.1	Schrittmotoren . . . . .	34
8.2.2	Schrittmotortreiber . . . . .	35
8.3	Funktionseinheit Kalibrierung . . . . .	37
8.3.1	Lichtschranke . . . . .	37
8.3.2	Gyroensor . . . . .	37
8.4	Platinen . . . . .	38
8.4.1	Abwärtswandler . . . . .	38
8.4.2	Schaltplan . . . . .	38
8.4.3	Layout . . . . .	40
<b>9</b>	<b>Code</b>	<b>42</b>
9.1	Motor . . . . .	42
9.1.1	Konstruktor . . . . .	42
9.1.2	Bewegen des Motors . . . . .	43
9.2	Lidar . . . . .	44
9.2.1	Konstruktor und Variablen . . . . .	44
9.2.2	Aufnehmen von Messdaten . . . . .	45
9.3	Steuerung . . . . .	46
9.3.1	Pin Definitionen und Initialisieren der Klassen . . . . .	47

9.3.2 Zusätzliche Funktionen . . . . .	48
9.3.3 Variablen deklaration und aufrufen von Funktionen . . . . .	49
9.3.4 Erstellen der Datei zum Speichern der Daten . . . . .	50
9.3.5 Schleifen der Steuerung . . . . .	51
<b>10 Auswertung und Darstellung mit Matlab</b>	<b>53</b>
10.1 Importieren und Zuordnen der Messwerte . . . . .	53
10.2 Umwandlung von Kugelkoordinaten zu kartesischen Koordinaten . . . . .	54
10.3 Darstellung der Messwerte . . . . .	56
<b>11 Validierung des Systems</b>	<b>60</b>
11.1 Genauigkeit des Systems . . . . .	60
11.2 Vergleich verschiedener Auflösungen . . . . .	63
11.2.1 Übersicht über die Dauer, Auflösung und Anzahl an Messpunkten .	63
11.2.2 Vergleich der unterschiedlichen Auflösungen . . . . .	64
11.3 Vergleich der Sensoren . . . . .	68
<b>12 Fazit/Zusammenfassung</b>	<b>70</b>
<b>13 Ausblick</b>	<b>72</b>
13.1 Hardware . . . . .	72
13.1.1 Platine . . . . .	72
13.1.2 Gyrosensor . . . . .	73
13.1.3 Schleifring . . . . .	73
13.1.4 Motortreiber . . . . .	73
13.1.5 Bedienfeld . . . . .	73
13.2 Software . . . . .	74
13.2.1 Steuerung mit Übergabeparameter . . . . .	74
13.2.2 Bedienfeld Statusausgabe und Steuerung . . . . .	76
13.2.3 Webinterface . . . . .	76
<b>Anhang</b>	<b>A</b>
Literatur . . . . .	A
Aufteilung der Kapitel . . . . .	C
Github Repository . . . . .	D

# Abkürzungsverzeichnis

<b>BSP</b>	Board Support Package
<b>LIDAR</b>	Light Distance and Ranging
<b>ToF</b>	Time of Flight
<b>3D</b>	Dreidimensional
<b>CAD</b>	Computer Aided Design
<b>NEMA</b>	National Electrical Manufacturers Association
<b>GPIO</b>	General Purpose Input Output
<b>APD</b>	Avalanche Photo Diode
<b>SPAD</b>	Single Photon Avalanche Diode
<b>csv</b>	comma separated values
<b>SPI</b>	Serial Peripheral Interface
<b>CNC</b>	Computerized Numerical Control
<b>UART</b>	Universal Asynchronous Receiver Transmitter
<b>I<sup>2</sup>C</b>	Inter-Integrated Circuit
<b>IC</b>	Integrated Circuit
<b>PSD</b>	Position Sensitive Detector
<b>LED</b>	Light Emissiting Diode
<b>GUI</b>	Graphical User Interface
<b>WLAN</b>	Wireless Local Area Network

# Abbildungsverzeichnis

2.1	Schematischer Aufbau einer Photodiode [20] ( $p^+$ starke p-Dotierung) . . . . .	3
2.2	Schematischer Aufbau einer APD [4] ( $p^+$ starke p-Dotierung, $p^- (\pi)$ schwache (intrinsische) p-Dotierung, $n^+$ starke n-Dotierung) (1 - Metallkontakte, 2 - Entspiegelung) . . . . .	4
2.3	Aufbau einer Position Sensitive Detector (PSD) [4] . . . . .	5
3.1	Time of Flight (ToF) Prinzip [8] . . . . .	7
3.2	Versuchsaufbau Phasenverschiebungsentfernungsmessung [22] . . . . .	10
3.3	Phasendifferenz einer Welle [7] . . . . .	11
3.4	Beispiel Triangulation . . . . .	13
5.1	Versuchsaufbau . . . . .	18
7.1	Oberer Aufbau der Mechanik . . . . .	25
7.2	Basis der Mechanik . . . . .	26
7.3	Rahmen . . . . .	27
7.4	Mechanik mit montierter Elektronik . . . . .	29
8.1	TF Mini . . . . .	31
8.2	Abhängigkeit der Genauigkeit des VL53L1X Sensors von der Messfrequenz	32
8.3	Schematische Beschriftung der A4988 Trägerplatine . . . . .	35
8.4	Schaltplan: Spannungsversorgung . . . . .	39
8.5	Schaltplan: Motortreiber . . . . .	39
8.6	Schaltplan: Schnittstellen . . . . .	40
8.7	Schaltplan: Leds und Kühler . . . . .	40
8.8	Platinenlayout . . . . .	41
10.1	Kugelkoordinaten . . . . .	55
10.2	Darstellung mit Linien . . . . .	57
10.3	Darstellung mit Asterisken . . . . .	58
10.4	Darstellung mit Punkten . . . . .	58

---

11.1 Grundriss des Testraums . . . . .	60
11.2 Grundriss des Testraums . . . . .	61
11.3 Vogelperspektive des Testraums . . . . .	62
11.4 Grafischer Vergleich der Grundrisse . . . . .	62
11.5 Niedrige Auflösung . . . . .	65
11.6 Mittlere Auflösung . . . . .	66
11.7 Hohe Auflösung . . . . .	66
11.8 Vogelperspektive niedrige Auflösung . . . . .	67
11.9 Vogelperspektive hohe Auflösung . . . . .	67
11.10TF MINI . . . . .	68
11.11Sensor VLX . . . . .	68
11.12Sensor VLX . . . . .	69
11.13Beispielbild . . . . .	69
11.14Beispielbild . . . . .	69

# Tabellenverzeichnis

5.1	Bauteilliste Versuch 1 . . . . .	17
5.2	Messgeräteliste Versuch 1 . . . . .	18
5.3	Bauteilliste Versuch 2 . . . . .	20
5.4	Messgeräteliste Versuch 2 . . . . .	20
8.1	Distanzmodi VL53L1X . . . . .	32
8.2	Tabellarischer Vergleich TF Mini und VL53L1X . . . . .	34
8.3	Schrittweite . . . . .	36
9.1	Beispieldatei . . . . .	50
11.1	Übersicht verschiedene Auflösungen . . . . .	64
A1	Aufteilung der Kapitel . . . . .	C
A2	Verweise auf Github Repository . . . . .	D

# Formelverzeichnis

2.1 Position des Lichtpunkts einer PSD . . . . .	5
3.1 Berechnung der Entfernung mittels Lichtlaufzeit . . . . .	8
3.2 Berechnung der minimalen und maximalen Zeit . . . . .	9
3.3 Zurückgelegte Strecke des Lichtstrahls . . . . .	10
3.4 Zurückgelegte Strecke des Lichtstrahls . . . . .	11
3.5 Beispielrechnung Messgenauigkeit . . . . .	12
3.6 Sinussatz . . . . .	13
3.7 Beispielrechnung Trigonometrie . . . . .	14
7.1 Berechnung Übersetzungsverhältnis [5] . . . . .	28
8.1 Berechnung maximaler Strom für Schrittmotoren [2] . . . . .	36
10.1 Umrechnung Kugelkoordinaten in kartesische Koordinaten . . . . .	55
11.1 Berechnung horizontiale Auflösung . . . . .	63
11.2 Berechnung vertikale Auflösung . . . . .	64

# Listings

9.1	Bibliotheken der Motor Klasse . . . . .	42
9.2	Konstruktor der Motor Klasse . . . . .	43
9.3	Funktion zum Bewegen des Motors . . . . .	44
9.4	Bibliotheken der Lidar Klasse . . . . .	44
9.5	Kostruktor der Lidar Klasse . . . . .	45
9.6	Funktion um Distanz vom LIDAR Sensor zu erhalten . . . . .	45
9.7	Bibliotheken zur Steuerung des Systems . . . . .	46
9.8	Initialisieren von Variablen und Klassen . . . . .	47
9.9	Funktionen für die Übersichtlichkeit des Codes . . . . .	48
9.10	Aufrufen von Funktionen und Variablen deklaration . . . . .	49
9.11	Erstellen der Datei zum Speichern der Daten . . . . .	50
9.12	Messen und Aufzeichnen der Entfernung . . . . .	51
10.1	Importieren und Zuordnen von .csv Dateien . . . . .	54
10.2	Umwandlung von Kugelkoordinaten zu kartesischen Koordinaten . . . . .	56
10.3	Darstellung der Messwerte . . . . .	56
10.4	Skalieren der Achsen . . . . .	59
13.1	Beispiel Aufruf einer Python Funktion mit Übergabeparametern . . . . .	74
13.2	Python Beispiel Funktion welche Übergabeparamenter akzeptiert und ausführt	75
13.3	Installation dnsmasq hostapd . . . . .	77
13.4	Konfiguration DHCP Server . . . . .	77
13.5	Konfiguration DHCP Server . . . . .	77
13.6	Konfiguration Interfaces . . . . .	77
13.7	Konfiguration Interfaces . . . . .	77
13.8	Konfiguration Interfaces . . . . .	78
13.9	Konfiguration Hostapd . . . . .	78
13.10	Konfiguration Hostapd . . . . .	78
13.11	Konfiguration Hostapd . . . . .	78
13.12	Konfiguration Hostapd . . . . .	79

---

13.13Konfiguration dnsmasq . . . . .	79
13.14Konfiguration dnsmasq . . . . .	79
13.15Konfiguration IPV4 . . . . .	79
13.16Konfiguration IPV4 . . . . .	79
13.17Konfiguration IPV4 . . . . .	80
13.18Konfiguration IPV4 . . . . .	80
13.19Konfiguration IPV4 . . . . .	80
13.20Starten der neu installierten Packages . . . . .	80
13.21Installation NodeJS . . . . .	80
13.22Installation NodeJS . . . . .	81
13.23Installation Flask . . . . .	81
13.24Installation Flask . . . . .	81
13.25Flask Beispielprogramm . . . . .	81

# 1 Einleitung

Auf Baustellen und in der Innenarchitektur müssen ständig Räume vermessen werden. Herkömmliche Methoden, wie manuelle Laserentfernungsmesser sind teils ungenau und beanspruchen viel Zeit. Zudem werden dabei nur einzelne Raummaße ermittelt und 2D Grundrisse erstellt. Die für viele Planungsschritte notwendigen 3D Raumpläne werden aus Zeit- und Kostengründen nur selten erstellt. Dadurch ist die Bearbeitung ortsfreier Projekte nur schwer oder gar nicht umsetzbar und häufig mit Planungsfehlern verbunden. Geräte, die das Erstellen von 3D Raumkarten ermöglichen, sind sehr kostenintensiv und meist komplex in der Anwendung. Daher werden meist Dienstleistungsunternehmen benötigt, die die 3D Raumkarten erstellen. Die damit verbundenen enormen Kosten führen dazu, dass diese Methodik für kleine bis mittelgroße Bauprojekte nicht wirtschaftlich ist.

Im Rahmen dieser Studienarbeit soll ein System entstehen, welches das Erstellen von 3D Raumkarten ermöglicht. Ziel ist es, ein möglichst kostengünstiges und benutzerfreundliches Gerät zu entwickeln. Dieses soll in einem Raum aufgestellt werden und nach abgeschlossenem Messvorgang den Raum durch eine Punktewolke in 3D abbilden. Zusätzliche Anforderungen an das System sind eine hohe Genauigkeit und eine berührungslose Messung mittels eines LIDAR-Sensors.

Diese entstandene Punktewolke soll dazu dienen, ortsfreie Planung zu erleichtern. In dem 3D Modell des Raums können beliebige Maße direkt abgelesen werden. Das Modell ist drehbar, stellt komplexe Konturen dar und ermöglicht jegliche Ansicht. Die 3D-Modellierung kann durch die frei wählbare Ansichten zusätzlich für virtuelle Immobilienpräsentationen und weitere Animationen verwendet werden.

Der erste Teil dieser Studienarbeit beschäftigt sich mit den Grundlagen der Laserentfernungsmessung. Im Anschluss daran folgt eine Machbarkeitsstudie, einen LIDAR Sensor mittels kostengünstiger Komponenten selbst entwerfen und zu bauen. Das Resultat dieser Machbarkeitsstudie bestimmt den verwendeten Sensor für das Projekt. Ist ein kostengünstiger Selbstbau eines Sensors möglich, der den Anforderungen entspricht, wird dieser verwendet. Ist dies nicht möglich, werden passende Sensoren ausgewählt und zugekauft.

Der direkte Vergleich der Sensoren erfolgt mit dem fertigen System.

Der Hauptteil der Arbeit beschäftigt sich mit der Konzipierung, dem Bau und den Tests des eigentlichen Systems. Während der Konzipierung und des Baus, wird das Projekt in drei Kategorien aufgeteilt.

Der Erste Teil ist die Mechanik, welche den Sensor in zwei Achsen drehen kann um einen kompletten dreidimensionalen Raum abtasten zu können. Der zweite Teil beschäftigt sich mit den elektronischen Komponenten und deren Verbindungen untereinander. Der dritte Teil beschäftigt sich mit der Programmierung sowohl der Ansteuerung der Komponenten aus Teil 2, als auch der Auswertung und Visualisierung der Daten.

Im Anschluss daran wird das System getestet und verschiedene Auflösungen, Darstellungsarten und Sensoren verglichen.

*TODO: Überarbeiten, Ablauf genauer beschrieben, z.b. Genau Auflösung?*

## 2 Grundlagen Elektronik

### 2.1 Photodioden

Um Licht zu detektieren werden meist Photodioden verwendet. Diese arbeiten nach einem relativ einfachen Prinzip. Eine p-n-Diode wird in Sperrrichtung betrieben, durch die Angelegte Spannung entsteht eine Sperrsicht. Wenn nun Photonen auf die offene, starke p-Dotierung treffen werden dort durch den Photoeffekt Ladungsträger erzeugt (Abbildung: 2.1). Wenn diese nun durch Diffusion bis zur Sperrsicht gelangen, driften die Ladungsträger entgegen der Sperrspannung in die jeweiligen Raumladungszonen, dies ist als Strom messbar. [20]

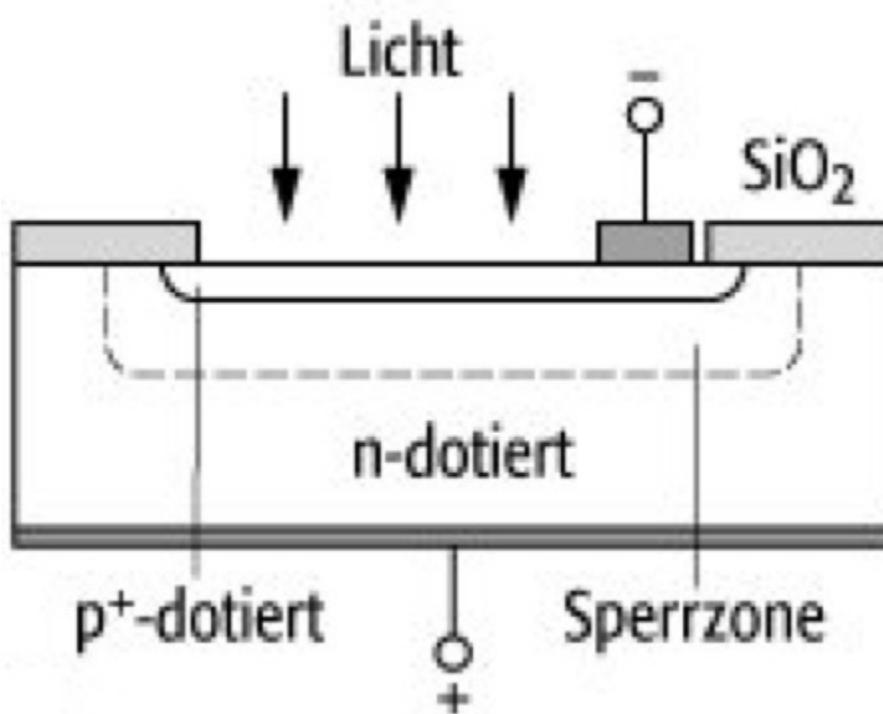


Abbildung 2.1: Schematischer Aufbau einer Photodiode [20] ( $p^+$  starke p-Dotierung)

Dieser Effekt tritt allerdings nur auf, wenn die Photonen eine Energie größer als die des Bandabstandes des verwendeten Halbleiters aufweisen. Hierbei ist zudem pro eintreffendem Photon nur ein sehr geringer Stromimpuls messbar, daher ist diese Art von Diode für LIDAR Anwendungen nicht brauchbar.

### 2.1.1 Avalanche Photo Diode (APD)

Um einzelne Photonen detektieren zu können wird eine spezielle Form der Photodiode verwendet. Die sogenannte APD. Die APD hat im Gegensatz zur herkömmlichen Photodiode zwei weitere Schichten. Hinzu zur n-Dotierten und stark p-Dotierten Schicht kommen nun eine schwach p-Dotierte (oder intrinsische) und eine "normal" p-Dotierte Schicht (Abbildung: 2.2).

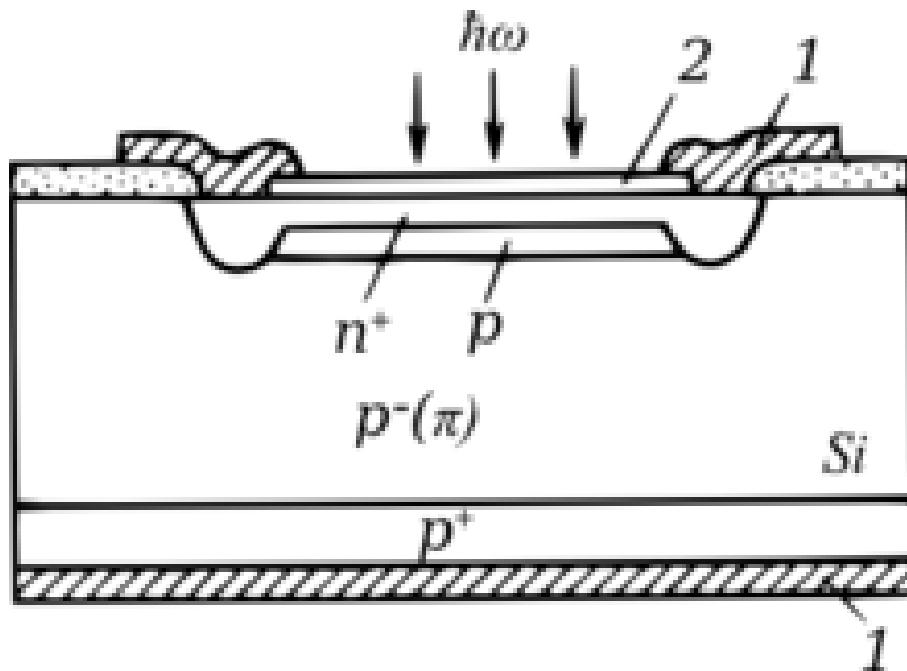


Abbildung 2.2: Schematischer Aufbau einer APD [4] ( $p^+$  starke p-Dotierung,  $p^-(\pi)$  schwache (intrinsische) p-Dotierung,  $n^+$  starke n-Dotierung) (1 - Metallkontakte, 2 - Entspiegelung)

Wenn Photonen nun in die  $\pi$  Zone gelangen, werden dort Landungsträger erzeugt, diese werden gleich wie bei der regulären Photodiode getrennt, Löcher wandern Richtung  $p^+$ -Zone und Elektronen Richtung  $n^+$ -Zone. Durch die stärker Dotierte p-Zone, und somit höhere Feldstärke, werden die Elektronen beschleunigt und es entsteht eine Stoßionisation. APD werden mit sehr hohen Sperrspannungen  $\sim 100V$ , nahe der Durchbruchspannung betrieben. [12]

Wenn die APD oberhalb der Durchbruchsspannung betrieben wird, setzt sich die Stoßionisation lawinenartig fort (Avalanche-Effekt) und es entstehen Verstärkungsfaktoren von einigen Millionen. APD welche speziell für den Betrieb oberhalb der Durchbruchsspannung ausgelegt sind werden auch Single Photon Avalanche Diode (SPAD) genannt. Mittels diesem Effekt kann man einzelne Photonen nachweisen, da jedes Photon einen kurzen detektierbaren Stromimpuls erzeugt. Bei der Anordnung vieler solcher SPADs in einem Array können viele einzelne Photonen präzise nachgewiesen werden. [16]

## 2.1.2 Lateral auflösende Photodiode

Die laterale auflösende Photodiode, auch PSD genannt, verwendet mehrere aneinander geschlossene Photodioden zur Bestimmung der Position eines Lichtpunktes.

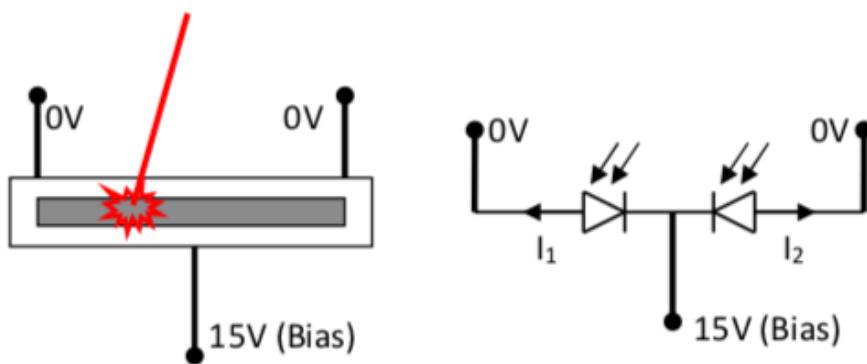


Abbildung 2.3: Aufbau einer PSD [4]

Wenn man nun jeweils den Photostrom der Dioden miteinander vergleicht kann man eine sehr präzise Auskunft über die Position des Lichtpunkts geben.

$$x = \frac{I_1 - I_2}{I_1 + I_2} \quad (2.1)$$

$I_1$  = Strom aus der linken Photodiode [A]

$I_2$  = Strom aus der rechten Photodiode [A]

$x$  = Relative Position des Lichtpunktes

Die Differenz der Ströme wird hier auf den Gesamtstrom Normiert, dies hat zur Folge, dass die Position unabhängig von der Intensität des Lichts wird.

Um diese Auswertung zu realisieren ist lediglich eine Transimpedanzverstärkerschaltung gekoppelt mit einer Komperatorschaltung nötig, mit welcher die Spannungen verglichen werden können.

Ein sehr großer Vorteil der PSD gegenüber anderer Methoden zur Feststellung der Position eines Lichtpunkts ist, dass die PSD innerhalb von Nanosekunden reagieren und das mit einer Sehr genauen Auflösung.[10]

## 2.2 Schrittmotoren

Beim Schrittmotor handelt es sich um einen Synchronmotor. Innerhalb des feststehenden Strators befindet sich ein drehend gelagerter Rotor. Wird ein Schrittmotor entsprechend angesteuert, dreht sich der Rotor um einen bestimmten Drehwinkel weiter. Durch mehrere Schritte kann der Rotor um jeden Drehwinkel, der einem Vielfachen des minimalen Drehwinkels entspricht, gedreht werden.

Man unterscheidet drei Bauformen von Schrittmotoren:

Der Reluktanz-Schrittmotor ist die älteste Bauweise von Schrittmotoren. Dabei besteht der gezahnte Rotor sowie der Strator aus weichmagnetischen Material. Durch Anlegen von Strömen bilden sich magnetische Felder aus und der Rotor dreht sich. Durch die fehlenden Permanentmagneten hat der Motor jedoch kein Rastermoment im ausgeschalteten Zustand. Beim Permanentmagnet-Schrittmotor besteht der Strator aus Weicheisen und der Rotor aus Permanentmagneten. Durch geschickte Bestromung des Strators, wird der Rotor immer so ausgerichtet, dass eine Drehbewegung entsteht. In dieser Bauform ist die Auflösung durch die limitierte Anzahl von Polen begrenzt.

Beim Hybridschrittmotor werden die Vorzüge der beiden genannten Motorarten vereint. Der Strator besteht aus gezahntem Weichmetall. Der Rotor besteht aus einem Permanentmagneten mit axialer Magnetfeldausrichtung. Darauf werden zwei fein gezackte, weichmagnetische Dynamobleche angebracht. Diese sind zueinander verdreht, sodass es zu einer Polteilung kommt und sich Süd- und Nordpole abwechseln. Der Hybridmotor zeichnet sich durch gutes Drehmoment und eine gute Auflösung aus.

*TODO: Quellen und Bilder einfügen, allgemeine Funktion beschreiben*

# 3 Grundlagen Laserentfernungsmessung

Um eine Entfernung zu einem Punkt mittels Licht zu bestimmen gibt es verschiedene Möglichkeiten, welche im folgenden Kapitel näher behandelt werden. Ein wichtiger Hinweis ist zudem, dass im Zusammenhang mit dem Thema LIDAR oftmals der Begriff 'Time of Flight (ToF)' fällt, dieser beschreibt allerdings nicht immer das direkt damit verbundene Verfahren, sondern allgemein die Entfernungsbestimmung mittels Licht.

## 3.1 Lichtlaufzeitmessung

### 3.1.1 Grundprinzip

Das Grundprinzip der Lichtlaufzeitmessung oder auch Time of Flight (ToF) (Abbildung: 3.1), bezieht sich auf die Zeit, welche ein ausgesandter Lichtimpuls benötigt bis er wieder am Sender eintrifft.

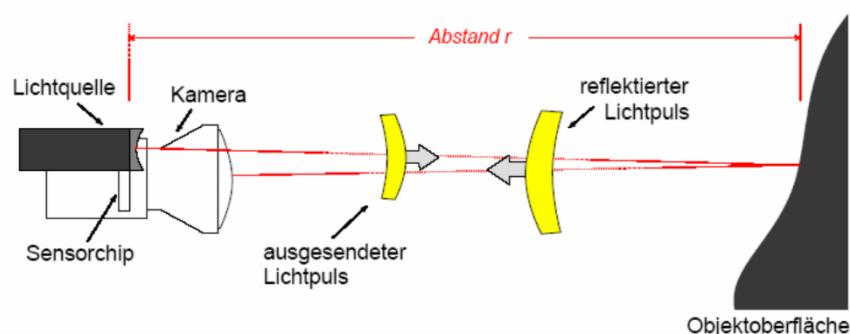


Abbildung 3.1: ToF Prinzip [8]

Dazu wird ein einzelner kurzer Lichtpuls von der Lichtquelle ausgesandt, welcher dann von der Oberfläche reflektiert wird und anschließend von einem Sensorchip wieder detektiert werden kann. Über die Zeitdifferenz zwischen aussenden und detektieren des Lichtimpulses

und die (halbe) Lichtgeschwindigkeit kann anschließend auf die Entfernung des getroffenen Punktes geschlossen werden. [9]

$$r = \frac{t_{diff}}{2} \cdot c \quad (3.1)$$

$r$  = Abstand zum getroffenen Punkt [m]

$t_{diff}$  = Zeitdifferenz zwischen aussenden und detektieren des Lichtpulses [s]

$c$  = Lichtgeschwindigkeit in Luft  $\left[ \frac{m}{s} \right]$

Ein großer Vorteil des ToF Verfahrens ist, dass durch die Reflexion von Partikeln in der Luft beispielsweise auf die Luftqualität oder die Luftfeuchtigkeit geschlossen werden kann. Dies macht sich am Sensor als vor der größten Reflexion (Oberfläche) auftretende kleine Reflexion bemerkbar.

### 3.1.2 Herausforderungen

Bei dieser Technologie entstehen allerdings einige Probleme, auf welche im Folgenden eingegangen wird. Generell sind alle Probleme welche bei ToF auftreten miteinander verknüpft, bzw. bedingen sich gegenseitig.

Das erste Problem welches Auftritt ist, dass nie das gesamte ausgesandte Licht zur Detektion zur Verfügung steht. Durch verschiedene Reflexionsgrade verschiedener Oberflächen und die generelle Streuung des Lichts bei auftreffen auf eine Oberfläche wir immer nur ein geringer Teil direkt zum Sensor zurückgeworfen. Daher sind hoch empfindliche Sensoren nötig um eine zuverlässige Detektion zu ermöglichen. SPAD sind für die Anwendung in einem ToF LIDAR System sehr gut geeignet, vor allem in einer Anordnung zu einem Array, da eine größere Sensorfläche mit gleichbleibender Genauigkeit realisiert werden kann, und somit eine größere Streuung des reflektierten Lichts abgedeckt werden kann. Allerdings ist das Detektieren des zurückgeworfenen Lichts nicht die einzige Herausforderung welche auftritt, denn mit steigender Distanz nimmt die Dämpfung des Lichtimpulses immer weiter ab. Eine Lösung dafür könnte sein eine Stärkere Lichtquelle zu verwenden, dies birgt allerdings große Sicherheitsrisiken und ist daher nur begrenzt möglich. Eine zweite Lösung ist, die maximale Messentfernung zu limitieren, allerdings birgt dies ein anderes Problem dieses und ein weiteres Problem, welches im Zusammenhang mit der minimalen Messentfernung steht, wird anhand eines Beispiels erläutert.

Man nehme an, es existiert ein fiktiver LIDAR Sensor mit folgenden Werten:

- Minimale Messentfernung  $d_{min} = 1\text{cm}$

- Maximale Messentfernung  $d_{max} = 1000m$
- Maximale Messfrequenz  $f_{max} = 150kHz$

Formel 3.1 kann umgestellt werden, um die Zeiten zu errechnen, welche das Licht für die minimale und maximale Messentfernung benötigt.

$$t = \frac{r \cdot 2}{c}$$

$$t_{min} = \frac{0,01 \cdot 2}{299,79 \cdot 10^6} = 667,13 \cdot 10^{-9} \quad (3.2)$$

$$t_{max} = \frac{1000 \cdot 2}{299,79 \cdot 10^6} = 6,6713 \cdot 10^{-6}$$

$r$  = Abstand zum getroffenen Punkt [m]

$t$  = Zeitdifferenz zwischen aussenden und detektieren des Lichtpulses [s]

$c$  = Lichtgeschwindigkeit in Luft  $\left[ \frac{m}{s} \right]$

Anhand des Beispiels kann man bereits erkennen, dass um die gewünschte minimale Messentfernung zu realisieren eine extrem schnelle Schaltung nötig ist, um das Aussenden und Empfangen innerhalb von  $667,13ns$  zu detektieren. Die maximale Messentfernung bringt in diesem Fall ein anderes Problem mit sich, welches noch nicht erwähnt wurde. Zwar ist dies nur ein fiktives Beispiel, allerdings muss das Problem trotzdem betrachtet werden. Der Sensor ist mit einer Maximalen Frequenz von  $150kHz \rightarrow 6,6667\mu s$  angegeben, allerdings kann mit dieser Frequenz die maximale Messentfernung nicht erreicht werden, da das Licht länger benötigt.

Durch dieses Beispiel wurde veranschaulicht, dass verschiedene Faktoren die Grenzen des Sensors festlegen. Die minimale Messentfernung wird definiert dadurch, wie schnell die Schaltung Aussenden und Empfangen detektieren kann. Die maximale Messentfernung wird von Leistung der Lichtquelle sowie Genauigkeit des Sensors definiert. Die maximale Messfrequenz hängt zusätzlich noch davon ab, wie schnell der Sensor die Daten zur Weiterverarbeitung z.B. an einer Universal Asynchronous Receiver Transmitter (UART) Schnittstelle bereitstellen kann.

## 3.2 Phasenverschiebung / Phasenmodulation

Das Phasenverschiebungsverfahren macht sich zu nutzen, dass bei einer ausgesandten Elektromagnetischen Welle die Phase immer größer wird bei steigender Entfernung. Durch

Aussenden verschieden Frequentierter Wellen kann dann die Phasenverschiebung der Wellen bestimmt werden und daraus die Entfernung[6].

### 3.2.1 Grundprinzip

Das Grundprinzip des Phasenverschiebungsverfahren basiert auf der Inferometrie. Die Inferometrie besagt lediglich, dass die Messergebnisse auf der Interferenz, also der Überlagerung mehrerer Wellen, beruht. Interferenzen treten bei verschiedensten Formen von Wellen auf, nicht nur bei Elektromagnetischen Wellen, sondern auch bei z.B. mechanischen Wellen wie Schall- oder Wasserwellen[14].

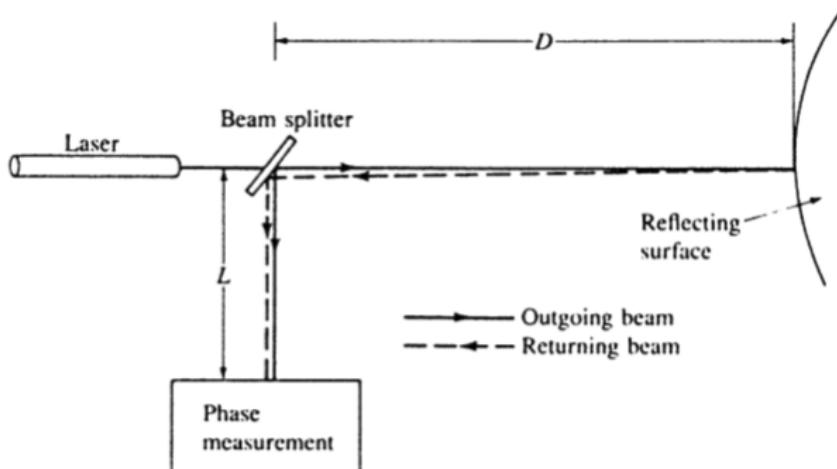


Abbildung 3.2: Versuchsaufbau Phasenverschiebungsentfernungsmeßung [22]

Bei einem Versuchsaufbau nach dem Prinzip in Abbildung 3.2, wird der vom Laser ausgesandte Lichtstrahl zuerst aufgeteilt, damit dieser dann als Referenzwert zur Bestimmung der Phasendifferenz verwendet werden kann. Der zweite Pfad des Lichtstrahls wird sich weiter in Richtung der Oberfläche bewegen, bis dieser schließlich von der Oberfläche reflektiert wird und durch einen Umlenkspiegel ebenfalls zur Bestimmung der Phasendifferenz verwendet wird.

Wenn die Distanz  $D = 0$  ist, kommen die beiden Lichtstrahlen gleichzeitig bei der Phasenbestimmung an. Bei einer Entfernung ungleich 0 ist die zurückgelegte Strecke des Lichtstrahls.

$$E = L + 2 \cdot D \quad (3.3)$$

$E$  = Zurückgelegte Strecke des Lichtstrahls [m]

$L$  = Abstand zwischen Spiegel und Phasenbestimmung [m]

$D$  = Abstand zwischen Spiegel und Oberfläche [m]

Da wie bereits erwähnt die Phase mit steigender Entfernung größer wird kann von dieser Phasenänderung auf die Wellenlänge geschlossen werden.

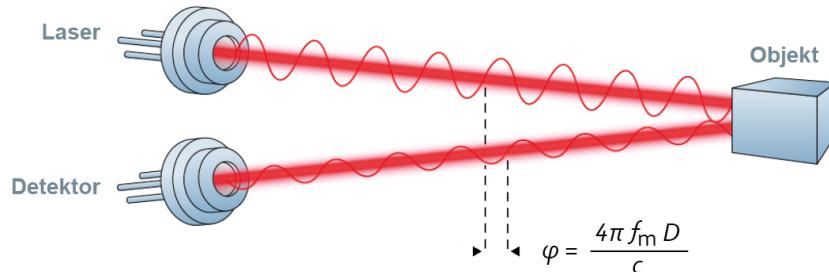


Abbildung 3.3: Phasendifferenz einer Welle [7]

Die Phase kann als ein Vielfaches der Wellenlänge  $\lambda = \frac{c}{f}$  und somit als Entfernung ausgedrückt werden.

$$\varphi = \frac{4 \cdot \pi \cdot D}{\lambda} \quad (3.4)$$

$$D = \frac{c \cdot \varphi}{4 \cdot \pi \cdot f_m}$$

$D$  = Abstand zwischen Laser und Oberfläche [m]

$\varphi$  = Phasendifferenz [°]

$\lambda$  = Wellenlänge [m]

$f_m$  = Modulationsfrequenz  $\left[ \frac{1}{s} \right]$

$c$  = Lichtgeschwindigkeit  $\left[ \frac{m}{s} \right]$

Allerdings bringt die Verwendung nur einer Wellenlänge zur Modulation des Lasers zur Bestimmung der Phasendifferenz einige Probleme mit sich. Zum einen wird die Gleichung 3.5 bei jedem Vielfachen  $n$  von  $360^\circ$  die gleiche Lösung ergeben, somit kann man die Distanz nicht eindeutig bestimmen. Zum anderen ist der Aufbau wie in 3.2 nur für wenige Anwendungen geeignet, daher wurde in Abbildung 3.3 bereits ein anderer besser nutzbarer Aufbau gezeigt.

Das Problem der Uneindeutigkeit kann gelöst werden, indem man den Lichtstrahl kontinuierlich mit mehreren Signalen mit unterschiedlichen Frequenzen moduliert wird. Anschließend wird für jede der ausgesandten Frequenzen die Phasendifferenz bestimmt. Dies bringt den großen Vorteil, dass durch Überlagerung der Ergebnisse eine Eindeutigkeit festgestellt werden kann und sich die Messgenauigkeit somit deutlich erhöht[22][23][18].

Einige der Vorteile, welche beim Phasenverschiebungsverfahren bestehen sind, dass die Bauteile um ein Vielfaches langsamer sein können, als beim ToF Verfahren. Dies ist darauf zurückzuführen, dass es prinzipiell egal ist bei welchem Nulldurchgang einer Welle die Phase bestimmt wird, da sich nur ein Vielfaches von  $360^\circ$  zur Phase addiert wird. Durch Verwendung mehrerer Modulationsfrequenzen kann trotzdem eine Eindeutigkeit gewährleistet werden.

Dazu kommt auch, dass mit Modulationsfrequenzen im Megahertz Bereich sehr hohe Messgenauigkeiten möglich sind, da diese von der höchsten Modulationsfrequenz abhängt.

$$\lambda = \frac{c}{f_m} = \frac{c}{150 \cdot 10^6} \approx 2 \quad (3.5)$$

$\lambda$  = Wellenlänge [m]

$f_m$  = Modulationsfrequenz  $\left[\frac{1}{s}\right]$

$c$  = Lichtgeschwindigkeit  $\left[\frac{m}{s}\right]$

Bei einer Wellenlänge von  $2m$  liegt der eindeutige Bereich zwar nur noch bei einem Meter, jedoch lässt sich die Phasendifferenz sehr genau bestimmen und somit ist eine höhere Genauigkeit möglich als beispielsweise bei einer Wellenlänge von  $150m$ .

### 3.2.2 Herausforderungen

Eine der größten Herausforderungen des Phasenverschiebungsverfahrens sind Störeinflüsse aus der Umwelt. Regen beispielsweise kann sich sehr störend auf die Messergebnisse auswirken. Ebenfalls kann keine Zusätzliche Information aus der Reflexion erlangt werden, wie dies beispielsweise beim ToF Verfahren möglich ist.

Die zum ToF Verfahren vergleichsweise wenigen Herausforderungen sind Grund dafür, dass das Phasenverschiebungsverfahren sehr weit verbreitet ist und in viele Anwendungen findet.

### 3.3 Triangulation

Beim Triangulationsverfahren wird sich wie der Name schon vermuten lässt die Trigonometrie zu nutzen gemacht.

#### 3.3.1 Grundprinzip

Wenn bei einem Dreieck zwei Punkte, deren Distanz zueinander und der Winkel zum Dritten Punkt bekannt ist, dann auch die Position des dritten Punktes bestimmt werden (Abbildung 3.4).

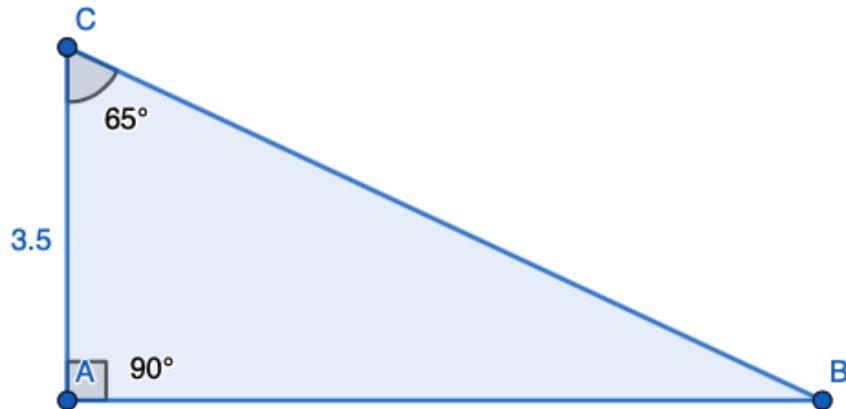


Abbildung 3.4: Beispiel Triangulation

Nun kann durch Anwendung bekannter Trigonometrischer Funktionen die Position des dritten Punktes, und damit die Entfernung, bestimmt werden.

$$\frac{c}{\sin(\gamma)} = \frac{b}{\sin(\beta)} = \frac{a}{\sin(\alpha)} \quad (3.6)$$

$a$  = Stecke BC [m]

$b$  = Strecke AC [m]

$c$  = Strecke AB [m]

$\alpha$  = Winkel an A [ $^\circ$ ]

$\beta$  = Winkel an B [ $^\circ$ ]

$\gamma$  = Winkel an C [ $^\circ$ ]

Für das gegebene Beispiel wäre dies:

$$\begin{aligned}\beta &= 180^\circ - \alpha - \gamma = 25^\circ \\ c &= \frac{b \cdot \sin(\gamma)}{\sin(\beta)} = 7.5 \\ a &= \frac{b \cdot \sin(\alpha)}{\sin(\beta)} = 8.28\end{aligned}\tag{3.7}$$

Da die Winkel bekannt sind ist nun bekannt, dass der Punkt B an  $x = 7.5$  und  $y = 0$  liegt. Somit wurde die Entfernung bestimmt.

Der Aufbau wie in Abbildung 3.4 kann auch mit Sender und Empfänger realisiert werden. Dazu wird angenommen, dass in Punkt A ein Sender, also Laser oder LED (nur für geringe Anforderungen) positioniert wird. In Punkt C wird ein PSD positioniert.

Wenn man nun eine Bikonvexe Linse vor das PSD positioniert wir das gesamte einfallende Licht gebündelt und als ein Punkt auf dem PSD messbar. Durch den Abstand des PSD zur Linse und die Position des Lichtpunkts auf dem PSD lässt sich dann der Winkel des einfallendes Lichts bestimmen.

Dieser Winkel kann dann wie beschrieben zur Bestimmung der Entfernung der reflektierenden Oberfläche verwendet werden.

Ein sehr großer Vorteil des Triangulationsverfahrens ist, dass es mit sehr preiswerten Bauteilen realisiert werden kann, da wie bereits erwähnt für Anwendungen mit sehr niedrigen Anforderungen eine Light Emitting Diode (LED) als Lichtquelle ausreicht. Ebenfalls sind die Photodioden nicht kompliziert oder teuer, anders als beispielsweise SPAD Dioden welche für ToF oder Phasenverschiebung Verfahren verwendet werden.

Zudem kann durch konstantes einschalten der Lichtquelle sehr gut eine Geschwindigkeit bestimmt werden, da die PSD innerhalb von wenigen Nanosekunden ihren Strom ändern und somit eine sehr schnelle Detektion eines sich bewegenden Objekts möglich ist. [3][10]

### 3.3.2 Herausforderungen

Durch Störeinflüssen wie beispielsweise anderes einfallendes Licht kann die Messung stark beeinflusst werden. Ebenfalls kann es gerade durch die Verwendung von preiswerten Bauteilen zu großen Toleranzen kommen wodurch die Messergebnisse stark schwanken können. Auch kann ähnlich wie beim Phasenverschiebungsverfahren keine Auskunft über weitere Parameter gegeben werden.

## 4 Stand der Technik

*TODO: Kapitel: Stand der Technik*

## 5 Machbarkeitsstudie

Ziel der Machbarkeitsstudie ist es, Informationen über die Schwierigkeit des Entwurfs und der Herstellung eines LIDAR Sensors mittels einer Photodiode zu erlangen.

### 5.1 Versuch 1: Test handelsüblicher Photodioden

In einem ersten sehr einfachen Aufbau wurde der Photostrom in einer Handelsüblichen Photodiode gemessen.

#### 5.1.1 Versuchsaufbau

##### Verwendete Bauteile

Tabelle 5.1: Bauteilliste Versuch 1

Bezeichnung	Bauteil (Beschreibung)
Photodiode	PD333-3C/HO/L2 EVL
Laserpointer	1mW, 630-680 nm
Taschenlampe	

## Verwendete Messgeräte

Tabelle 5.2: Messgeräteliste Versuch 1

Art des Messgeräts	Bezeichnung
Digitalmultimeter	Agilent
DC Spannungsquelle	

## Aufbau

Wie in Kapitel 2.1 beschrieben müssen Photodioden mit einer Sperrspannung betrieben werden, um einen möglichst großen Photostrom messen zu können, daher wurde die Photodiode mit einer Sperrspannung von 5V betrieben. Das Digitalmultimeter wurde in Reihe zur Photodiode geschlossen um den entstehenden Photostrom messen zu können. Um direkt einen möglichst guten Eindruck der späteren Anwendung zu bekommen wurde direkt ein Testaufbau verwendet, in welchem das Licht vom Laserpointer reflektiert und danach von der Photodiode detektiert wird. Dazu wurde die Photodiode direkt neben dem Laserpointer positioniert (Abbildung: 5.1).

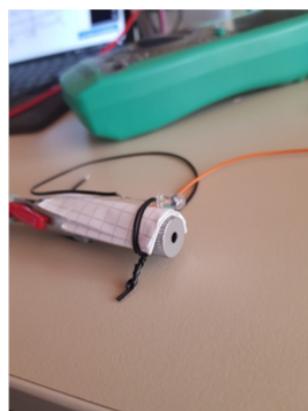


Abbildung 5.1: Versuchsaufbau

## Messungen

Insgesamt wurden vier Messungen durchgeführt.

1. Photostrom im nicht abgedunkelten Raum
2. Photostrom im abgedunkelten Raum

3. Photostrom bei direktem Beleuchten mit einer Taschenlampe
4. Photostrom bei direktem Beleuchten mit einem Laserpointer
5. Photostrom bei Reflexion von einem Laserpointer

Diese ersten vier Messungen sollen einen ersten Überblick über die Stärke des Photostroms geben und die Störeinflüsse des Umgebungslicht näher betrachten. Zudem kann über die fünfte Messung und die Variation des Abstandes der Reflexionsfläche Information darüber erlangt werden, wie groß die relative Änderungen des Photostroms bei relativer Änderung des Abstandes ist.

### 5.1.2 Beobachtung

Die größte Beobachtung, welche bei den Messungen gesehen werden konnte ist, dass der Photostrom nur im Bereich von wenigen  $\mu A$  liegt.

1. Nicht abgedunkelter Raum:  $15\mu A$
2. Abgedunkelter Raum:  $3 - 5\mu A$  (sogenannter Dunkelstrom)
3. Lichteinstrahlung Taschenlampe:  $80\mu A$
4. Lichteinstrahlung Laserpointer:  $200\mu A$

Diese ersten vier Messungen zeigten bereits, dass die Auftretenden Photoströme nur sehr gering sind und daher zusätzlich zu einer sehr schnellen Auswertelektronik (Messung nach dem ToF Prinzip) zudem eine sehr genaue Verstärkerschaltung nötig ist um gute Messergebnisse zu erzielen.

5. Bei Änderung des Abstandes der Reflexionsfläche in Messung fünf kann festgestellt werden, dass ein Abstand bis ca.  $7\text{cm}$  detektiert werden kann. Die relative Änderung des Photostroms beträgt dabei  $\frac{0,2 \mu A}{7 \text{ cm}}$

### 5.1.3 Erkenntnis

Die größte Erkenntnis aus den Messungen des ersten Versuchs ist, dass Störende Lichteinflüsse so gut wie möglich eliminiert werden müssen. Dies kann z.B. durch Filter, welche nur das vom Laser ausgesandte Licht durchlassen, realisiert werden.

Ein Weiterer Punkt welcher festgestellt wurde ist, wie bereits schon erwähnt wurde, dass eine sehr Präzise Schaltung zu Feststellung der Photoströme nötig ist, da die Änderungen

in sehr geringen Bereichen geschehen. Eine Lösung für dieses Problem könnte eine Verstärkerschaltung aus hochpräzisen Operationsverstärkern sein.

Der Dritte Punkt welcher in folgenden Messungen betrachtet werden sollte ist, dass eine Diode mit einer höheren Empfindlichkeit gegenüber der Lichteinstrahlung hat. Da wie beobachtet wurde die Reflektierte Lichtmenge sehr gering ist. Hier könnte beispielsweise eine Avalanche Photodiode verwendet werden.

## 5.2 Versuch 2: Weiterer Test handelsüblicher Photodioden

### 5.2.1 Versuchsaufbau

#### Verwendete Bauteile

Tabelle 5.3: Bauteilliste Versuch 2

Bezeichnung	Bauteil (Beschreibung)
Photodiode	BPW34
Laserpointer	1mW, 630-680 nm
Karton	
Lupe	

#### Verwendete Messgeräte

Tabelle 5.4: Messgeräteliste Versuch 2

Art des Messgeräts	Bezeichnung
Digitalmultimeter	Agilent
DC Spannungsquelle	

## Aufbau

In die Seite des Kartons wurde je ein Loch für Laserpointer und Photodiode geschnitten und diese jeweils darin Angebracht. Im Karton wurde eine Reflexionsfläche im Abstand von  $7\text{cm}$  aufgestellt und der Karton wurde verschlossen. Die Diode wurde mit einer Sperrspannung von  $5\text{V}$  betrieben und das Digitalmultimeter in Reihe zur Photodiode geschaltet um analog zu Versuch 1 den entstehenden Photostrom zu messen.

## Messungen

1. In der ersten Messung wurde die Photodiode zusammen mit dem Laserpointer im Dunklen getestet und der Photostrom im Dunkeln sowie bei Betätigung des Laserpointers gemessen.
2. In einer zweiten Messung wurde vor der Photodiode eine Lupe Positioniert, sodass das Licht auf die Photodiode gebündelt wird. Es wird erwartet, dass durch die Lupe ein größerer Photostrom entsteht.

### 5.2.2 Beobachtung

1. Zunächst ist zu Beobachten, dass im Dunkeln ein Photostrom von  $0,3\mu\text{A}$  durch die Photodiode fließt. Bei Betätigung des Laserpointers steigt der Photostrom um  $0,2\mu\text{A}$  an. Wenn der Abstand Reflexionsfläche zu Laserpointer und Diode vergrößert wird, sinkt der Stromanstieg bei Betätigung des Laserpointers noch weiter ab. Diese Beobachtungen sind Analog zu den Beobachtungen aus Versuch 1.
2. Bei Platzierung der Lupe vor der Photodiode ist ein stärkerer Stromanstieg bei Betätigung des Laserpointers zu Beobachten. Allerdings kann ist dieser Aufbau sehr schwer einstellbar. Hauptgrund dafür ist, dass der Laserpointer nun nichtmehr orthogonal zur Kartonwand aufgestellt werden kann, sondern in einem Winkel positioniert werden muss, damit die Reflexion möglichst exakt im Brennpunkt der Lupe positioniert ist. Dann kann auch der größtmögliche Stromanstieg beobachtet werden.

### 5.2.3 Erkenntnis

Die Erkenntnis aus dem zweiten Versuch ist, dass eine Optik vor der Photodiode eine größere Empfindlichkeit ermöglicht. Allerdings ist diese Optik auch sehr schwer einzustellen, da sich Einfallsinkel des Lichts mit steigender Entfernung kontinuierlich ändern. Zudem wurde Beobachtet, dass ein Dunkler Raum lediglich den Gesamtstrom reduziert, der Anstieg des Stroms bei Betätigung des Laserpointers bleibt gleich.

Die Bisherigen versuche beruhen auf dem ToF Prinzip, da wir ständig versuchen eine Flanke des Photostroms zu erkennen. Ein Problem welches dabei bisher noch nicht betrachtet wurde ist wie im Kapitel 3.1.2 beschrieben, dass bei geringer Distanz die Lichtlaufzeit im Bereich weniger  $ns$  liegt und daher eine sehr schnelle Auswertelektronik nötig ist. Daher wird für weitere Test eine wie in Kapitel 2.1.1 APD benötigt um wenige reflektierte Photonen besser detektieren zu können.

## 5.3 Weiteres Vorgehen

Wie bereits beschrieben sind weitere Versuche nur mit einer APD sinnvoll. Da diese allerdings meist in Preisspektren  $> 100\text{€}$  liegen wurden die Versuche eingestellt.

### 5.3.1 Recherche

Da es in unserem Unternehmen einige Abteilungen aus verschiedensten Bereichen gibt, welche sich mit Lserenfernungsmessung beschäftigen war der nächste Schritt Kontakt mit diesen Abteilungen aufzunehmen und die Pläne einen Eigenen LIDAR Sensor zu entwickeln zu diskutieren.

Die Erkenntnis welche aus diesen Gesprächen gewonnen werden konnte ist, dass ohne APD keines der drei in Kapitel ?? vorgestellten Messverfahren realisierbar ist. Zudem wurden Hinweise und Kritikpunkte an dem geplanten vorgehen geäußert und wichtige Hinweise zur generellen Realisierung des Gesamtsystems angebracht. So zum Beispiel, dass bei einem LIDAR Sensor nicht nur wichtig ist, was die minimale und maximale Messdistanz ist, sondern auch mit welcher Frequenz der Sensor diese Messergebnisse reproduzieren kann. Dies hat großen Einfluss auf Messgenauigkeit und Messdauer und ist daher ein weiterer Punkt welcher bei der Auswahl eines geeigneten Sensors betrachtet werden muss.

## 6 Matlab Modell

Zur Konzeptionierung des Systems und zur Auswahl der benötigten Komponenten müssen einige Vorüberlegungen angestellt werden. Die geforderte Auflösung und Genauigkeit des Lidar-Systems, sowie die Maximalzeit für das Erstellen der Punktewolke sind ausschlaggebende Parameter für die Komponentenauswahl.

Die gefordete Genauigkeit sowie der vordefinierte Standardraum zur späteren Vermessung definieren hauptsächlich die Anforderungen an den Lidar Sensor. Bei gegebener Maximalzeit für einen Scan, muss zusätzlich die Messfrequenz des Sensors dementsprechend hoch sein. Die Auflösung bestimmt die minimale erreichbare Schrittweite der Schrittmotoren. Dabei muss die nicht gleichmäßige Messpunkteverteilung an einer Wand berücksichtigt werden.

Das mittig im Raum aufgestellte Lidar-System nimmt eine Punktewolke des Raumes auf. Dazu soll sich der Sensor für jede Messung in zwei Achsen um einen vordefinierten Winkel weiterbewegen. Dies führt dazu, dass die Punkteverteilung trotz eines gleichbleibenden Winkels nicht homogen bleibt.

Dies Bsp nur Hotizontal:

Um die geforderte Auflösung auch noch an den am weitesten vom Lidar System entfernte Stellen zu erreichen, wird ein Matlab Modell erstellt. Bei diesem können Parameter..... eingestellt werden und man erhält die Punkteverteilung der Messung exemplarisch für eine Wand.

Matlab Modell einer Wand:

Schwierigkeiten: Kompromiss finden, Punkte Zentral davor und Eindeutigkeit eines Punkes, bzw Zuordnung zu einer Wand, Decke

# 7 Mechanik

## 7.1 Anforderungen

Damit ein 3D Abbild eines Raumes erstellt werden kann, ist es erforderlich, dass dieser möglichst leicht in mindestens zwei Achsen bewegen werden kann. Deshalb muss im Rahmen dieses Projekts eine geeignete Mechanik entworfen werden, welche es ermöglicht, den Sensor auf zwei getrennt voneinander steuerbaren Achsen beliebig positionieren zu können. Damit eine solche Mechanik entworfen werden kann müssen zuerst einige Rahmenbedingungen geklärt werden. Beispielsweise sollten die Motoren welche die Mechanik später antreiben vorher spezifiziert sein und die maximale Größe des Sensors bekannt sein. Natürlich sollte die Mechanik auch so entworfen werden, dass diese dann auch in der Praxis umgesetzt werden kann.

Zur besseren Visualisierung und um genaue Zeichnungen anzufertigen wurde ein Computer Aided Design (CAD) Zeichenprogramm verwendet.

## 7.2 Entwurf

Der gesamte Aufbau lässt sich in drei große Teile unterteilen. Einen oberen Aufbau, welcher das Kippen des Sensors übernimmt und einen Motor halten muss. Die Basis, welche sich um 360° Drehen lassen soll. Und den Rahmen, welcher die Steuerung und den zweiten Motor enthält.

### 7.2.1 Oberer Aufbau

Für den oberen Aufbau der Mechanik gab es mehrere Voraussetzungen. Zuerst soll die gesamte Mechanik so funktionieren, dass der Sensor möglichst genau im Ursprung der Dreh- und Kippachse liegt, um spätere komplizierte Umrechnungen der Punktewolke zu verhindern. Dazu soll der Aufbau möglichst leicht und klein sein, damit die Beschleunigung

Masse und die damit verbundenen Trägheitskräfte möglichst gering sind, damit unnötige Belastungen auf die Motoren vermieden werden. Außerdem müssen alle Leitungen, welche in dieser Aufbaute benötigt werden 360° Drehbar sein, weshalb ein sogenannter Schleifring unumgänglich ist.

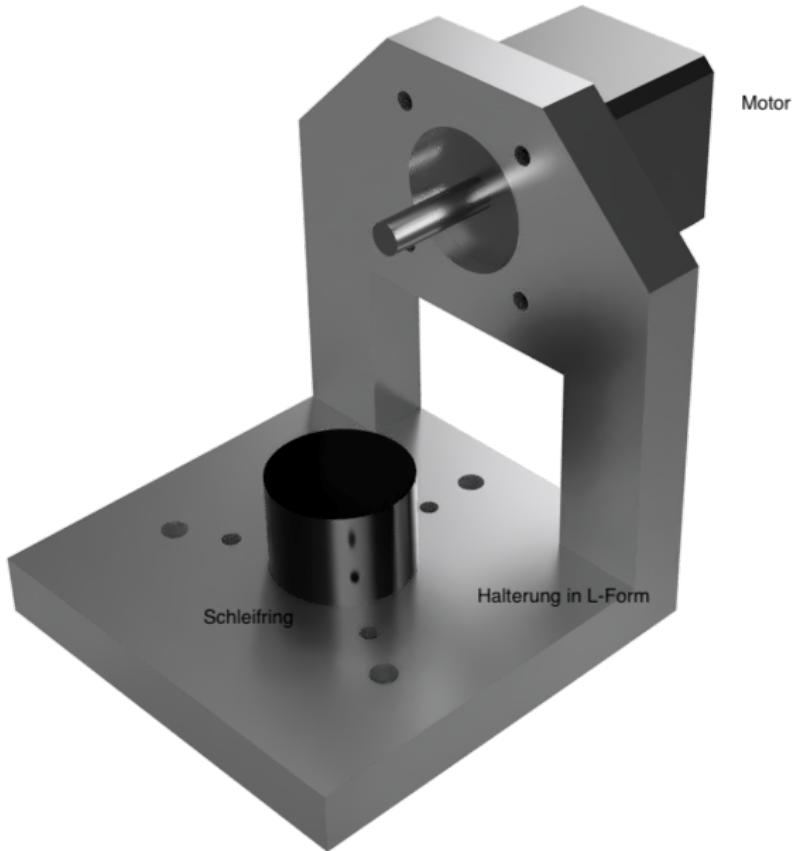


Abbildung 7.1: Oberer Aufbau der Mechanik

Der Motor welcher in Abbildung 7.1 zu sehen ist, ist von der National Electrical Manufacturers Association (NEMA) genormt und hat den Namen NEMA 11, die 11 verweist hierbei auf die Baugröße in diesem Fall 1,1" was ca. 28mm entspricht [15]. Außerdem ist in der Abbildung der Schleifring zu sehen, welcher später dazu dienen wird, dass alle Kabel des Oberen Aufbaus um 360° Drehbar sind.

Die Halterung in L-Form besteht aus zwei Teilen, welche aneinander Geschraubt werden. Ein horizontales Teil, die Grundplatte, welche den Schleifring und die Verbindung zu den weiteren Teilen sicherstellt. Und ein vertikales Teil, welches den NEMA 11 Motor in einer Vertiefung hält.

In Abbildung 7.1 fehlt allerdings ein weiteres Bauteil. Auf der Welle des Motors wird eine

weitere Platte montiert, worauf später der LIDAR Sensor montiert wird. Zur besseren Übersicht wurde in der gezeigten Ansicht auf diese Platte verzichtet.

### 7.2.2 Basis

Die Basis stellt die Verbindung zwischen dem Oberen Aufbau und dem Rahmen dar. Die Basis ist die komplexeste Baugruppe der gesamten Mechanik, da sie den Antrieb und die Lagerung des Oberen Aufbaus übernimmt.

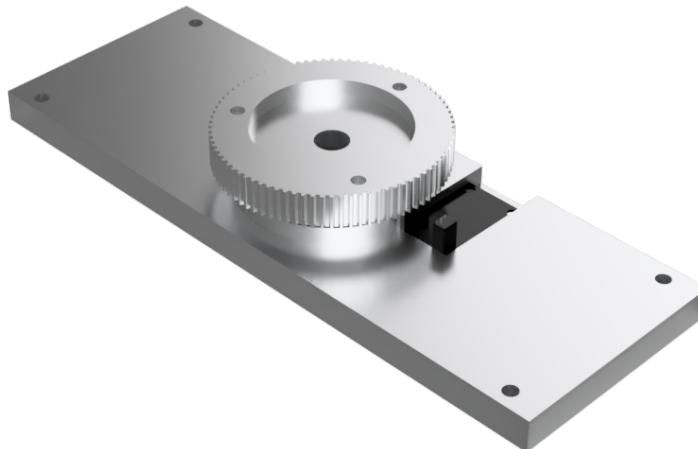


Abbildung 7.2: Basis der Mechanik

Um die Lagerung herzustellen wird ein großes Kugellager mit einem Innendurchmesser von  $22mm$  in die Verbindungsplatte (Abbildung: 7.2) eingepresst. Der große Innendurchmesser des Kugellagers ist erforderlich, damit die Kabel durch dieses hindurch geführt werden können. Der Antrieb des Oberen Aufbaus wird durch eine Zahnriemenscheibe hergestellt. Diese ist nach DIN 7721-2 T2,5 [5] entworfen da in dieser Anwendung eine große Anzahl an Zähnen gefordert ist, um eine höhere Winkelauflösung zu erhalten, wird diese Platte 3D gedruckt werden. Mit dem gewünschten mindest Durchmesser der Platte ergibt sich ein Umfang des Zahnrades von  $66,3mm$  und eine Zahlanzahl von 84 [5]. Um die Zahnriemenscheibe mit dem Kugellager zu verbinden wird eine Adapterplatte verwendet, welche innen in das Kugellager eingepresst wird und anschließend mit Zahnriemenscheibe und mit dem Oberem Aufbau verschraubt. Diese Adapterplatte hat ein durchgängiges Loch um die Kabel heraus zu führen. Zudem sitzt die Adapterplatte vertieft in der Zahnriemenscheibe, um die Baugröße kompakt zu halten und einen Formschluss zu erzeugen. Das letzte Bauteil

der Basis ist die Lichtschranke welche zur Positionierung dient. Diese Lichtschranke sitzt vertieft in der Basisplatte, damit der sich drehende Teil darüber passt. Durch einen Zapfen an der 3D gedruckten Zahnriemenscheibe wird die Lichtschranke ausgelöst.

### 7.2.3 Rahmen

Die dritte Baugruppe der Mechanik ist der Rahmen (Abbildung 7.3). Dieser dient hauptsächlich dazu eine stabile Befestigungsmöglichkeit für die Basis und den oberen Aufbau zu gewähren und die gesamte Elektronik zu ordnen. Zudem dient der Rahmen als Befestigungspunkt für den zweiten Motor. Der zweite Schrittmotor ist nach NEMA 17 genormt mit einem Außenmaß von ca  $41mm$ . Dieser wird über einen Zahnriementrieb den gesamten oberen Aufbau um  $360^\circ$  Drehen.

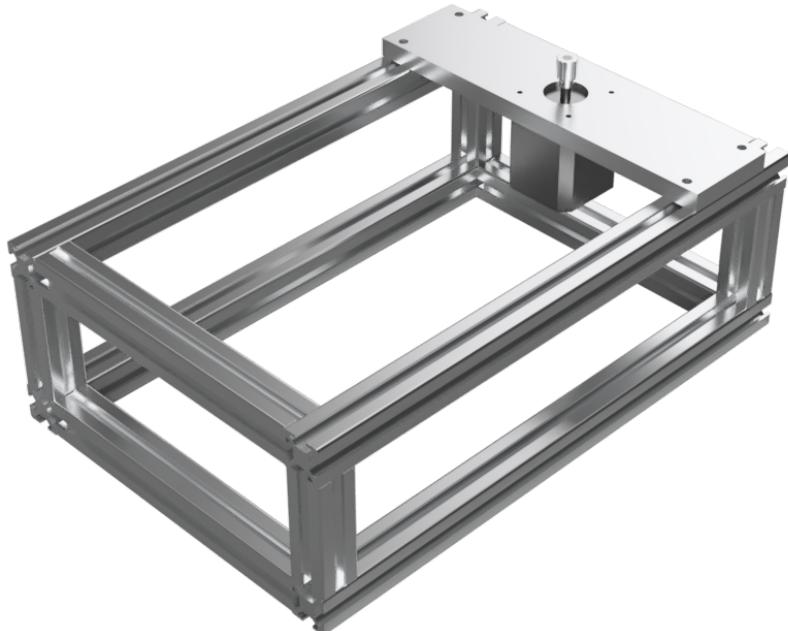


Abbildung 7.3: Rahmen

Um das obere Ende der Welle des zweiten Schrittmotors auf die selbe Höhe wie die Oberkante der Zahnriemenscheibe zu bringen ist eine weitere Halterung erforderlich. Auf der Motorwelle des NEMA 17 Motors sitzt eine weitere Zahnriemenscheibe, allerdings in einem deutlich kleineren Durchmesser ( $10,6mm$  Außendurchmesser, 14 Zähne). Durch die

beiden Verwendeten Zahnriemenscheiben ergibt sich ein Übersetzungsverhältnis welches im folgenden Berechnet wird:

$$i = \frac{z_a}{z_e} = \frac{84}{14} = 6 \quad (7.1)$$

$i$  = Gesamtübersetzungsverhältnis

$z_a$  = Zähnezahl getriebene Scheibe

$z_e$  = Zähnezahl treibende Scheibe

Durch das erhöhte Übersetzungsverhältnis wird eine Verlangsamung der Drehbewegung erreicht. Dies ermöglicht eine noch exaktere Auflösung, um eine komplette Drehung des Systems zu erreichen sich der NEMA 17 Motor sechs mal Drehen muss, daher werden auch sechs mal so viele Schritte pro Drehung benötigt.

Außerdem wird für den gesamten Rahmen ein Aluminiumprofil mit Nutensteinen verwendet. Dies ermöglicht das herstellen der benötigten Spannung auf dem Riemen welcher das System dreht. Zudem kann durch die Nuten im Aluminiumprofil einfach eine Bodenplatte zur Montage von Platine und Raspberry Pi eingesetzt werden.

## 7.2.4 Materialliste

*TODO: Materialliste*

## 7.3 Umsetzung

Nachdem die Zeichnungen von allen Bauteilen angefertigt und überprüft wurden, konnte mit der Herstellung der einzelnen Bauteile begonnen werden. Fast alle selbst konstruierten Bauteile wurden in Handarbeit aus Aluminium gefertigt, dabei wurde durch Fräsen, Drehen und Bohren die gewünschte Form erreicht. Da wie bereits erwähnt die Basisplatte das Komplexeste Bauteil ist, wurde diese extra von einer Computerized Numerical Control (CNC) Fräse gefertigt, ebenfalls die Adapterplatte wurde CNC Gedreht, damit die Passform des Kugellagers erreicht wird und die Bauteile perfekt eingepresst werden können. Die Zahnriemenscheibe, welche auf der Basis montiert wird, wurde Dreidimensional (3D) gedruckt, da ein herkömmliches Fertigungsverfahren mit unseren Mitteln nicht möglich gewesen wäre und dies zudem ohnehin sehr kompliziert und aufwändig ist.

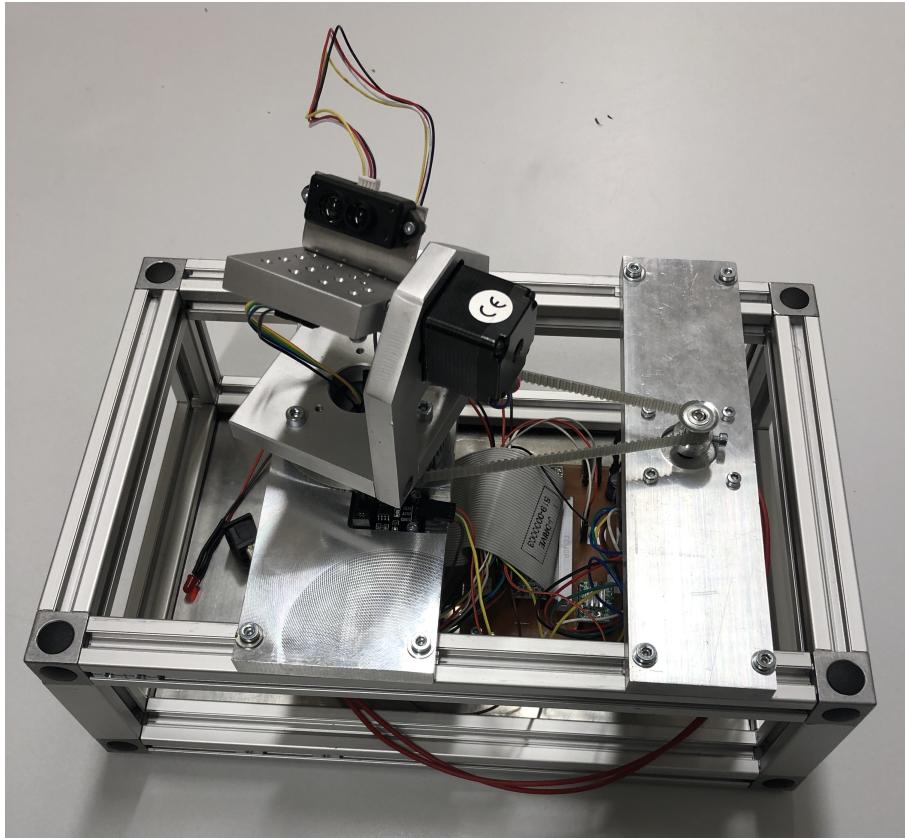


Abbildung 7.4: Mechanik mit montierter Elektronik

Nach Fertigstellung aller Einzelteile kann die Mechanik Zusammengebaut werden und die Elektronik eingebracht werden (Abbildung 7.4). Dabei wurden noch einige kleinere Teile gefertigt, von welchen kein extra CAD Modell erstellt wurde. So wurde eine Halterung für den LIDAR Sensor TF-Mini (Kapitel 8.1.1) aus einem Blech gebogen und gebohrt und eine Platte zugeschnitten und gebohrt, welche in den unteren Teil des Rahmens eingesetzt werden kann, damit darauf die Platine und der Raspberry Pi montiert werden kann. Nachdem auch diese kleineren Teile gefertigt wurden, konnte die Mechanik final zusammengebaut werden. Dabei wurden die Motoren an ihren vorgesehenen Plätzen montiert und die einzelnen Komponenten zusammengesetzt.

# 8 Hardware

Im folgenden Kapitel werden die elektronischen Hardwarekomponenten, sowie deren Verbindungen untereinander vorgestellt.

Die einzelnen Komponenten lassen sich in drei große Funktionsbereiche klassifizieren:

Der erste Bereich beinhaltet die Distanzbestimmung. Diese wird durch den LIDAR-Sensor realisiert.

Der zweite Funktionsbereich beschäftigt sich mit dem Ausrichten des Sensors in zwei Achsen. Die Komponenten sind zwei Schrittmotoren und die damit verbundene Ansteuerung durch Motortreiber.

Die automatisierte Kalibrierung stellt den dritten Funktionsbereich dar. Dabei wird über eine Lichtschranke die horizontale Ausrichtung des Sensors bei jedem Start der Anwendung auf eine vordefinierte Ausgangsposition gesetzt. Dasselbe wird durch einen Gyrosensor für die vertikale Ausrichtung ermöglicht.

Als Rechen- und Steuereinheit für das gesamte System wird ein Raspberry Pi verwendet. Dieser bietet sich aufgrund des geringen Preises, der vielen General Purpose Input Output (GPIO) Pins und der Unterstützung aller benötigten Datenübertragungsprotokolle an. Zudem reicht die Rechenleistung für das Ansteuern aller Komponenten, sowie für das Auswerten und Speichern der Messdaten aus.

## 8.1 Funktionseinheit Distanzbestimmung

Bei der Auswahl des LIDAR-Sensors spielen Preis, Verfügbarkeit, Messfrequenz, Auflösung, Genauigkeit und messbare Entfernung eine Rolle. Im Folgenden werden zwei ausgewählte Sensoren vorgestellt.

### 8.1.1 TF Mini LIDAR

Als Hauptsensor wird ein „TF Mini LIDAR“ von dem Hersteller „Seeedstudio“ verwendet. Dieser misst Entfernung mit auf dem Prinzip der Phasendifferenz wie in Kapitel 3.2 beschrieben.

Der Arbeitsbereich ist zwischen 30 cm und 1200 cm mit einer Auflösung von 1 cm. Bei Entfernungen kleiner als 600 cm beträgt die Messgenauigkeit 1%. Zwischen 600 cm und 1200 cm 2%. Die Messfrequenz beträgt maximale 100 Hz. Der Sensor arbeitet mit einer Versorgungsspannung von 4,5 V – 6 V bei einem durchschnittlichen Stromverbrauch von 120 mA. Das Kommunikation läuft über eine UART Schnittstelle mit einer Logikspannung von 3,3 V. [19]

Dieser Sensor wurde ausgewählt, da das Preis-Leistung Verhältnis und die Verfügbarkeit sehr gut ist. Zudem reicht der messbare Bereich für den zuvor definierten Standardraum aus. Die Kommunikation über UART ist mit dem Raspberry Pi realisierbar. Sensoren mit einer deutlich höheren Messfrequenz sind um ein Vielfaches teurer, weshalb aufgrund der Anforderung eines kostengünstigen Systems die Messfrequenz von 100 Hz ausreichend sein muss. Zusätzlich wird dadurch nicht die Qualität des Ergebnisses beeinflusst. Die niedrigere Messfrequenz beeinträchtigt nur die Geschwindigkeit, mit der ein Raum vermessen werden kann.

Der Sensor ist 42 x 15 x 16 mm groß, wodurch er sehr gut auf der dafür vorgesehenen Aluminiumhalterung montiert werden kann.



Abbildung 8.1: TF Mini

## 8.1.2 VL53L1X

Als alternativer, kostengünstigerer Sensor wird der ToF-Sensor VL53L1X verwendet. Der Sensor nutzt das Prinzip der Lichtlaufzeitmessung. Es können Distanzen von bis zu 400 cm mit einer maximalen Frequenz von 50 Hz gemessen werden. Die Kommunikation erfolgt über Inter-Integrated Circuit (I<sup>2</sup>C).

Die Versorgungsspannung beträgt zwischen 2,6 und 3,5 V. Sowohl der Erfassungswinkel als auch der interessante Entfernungsbereich kann softwaretechnisch eingestellt werden. [11]

Der Arbeitsbereich ist von dem eingestellten Distanzmodus abhängig. Man kann wie in Tabelle 8.1 dargestellt zwischen drei Modi auswählen. Die Tabelle zeigt zudem die maximal messbare Distanz für den jeweiligen Modus in Abhängigkeit von dem Umgebungslicht. Der Modus für kurze Distanzen ist Umgebungslicht unempfindlich. Bei den Modi für mittlere bis hohe Distanzen reagiert der Sensor sehr stark auf Umgebungslicht. Die maximal zu messende Distanz beträgt bei hohem Umgebungslicht nur noch ca. 75 cm.

Tabelle 8.1: Distanzmodi VL53L1X

Distanz Modus	max. Distanz (abgedunkelt)	max. Distanz (Umgebungslicht)
Kurz	136 cm	135 cm
Mittel	290 cm	76 cm
Lang	360 cm	73 cm

In der späteren Anwendung wird der Modus für große Distanzen benötigt. Daher sollten während der Messung potentielle Fehlerquellen durch Umgebungslicht vermieden werden.

Die Genauigkeit der Messung hängt wie in Abbildung 8.2 zu sehen von der Messfrequenz ab. Die Abbildung zeigt das Verhalten bei unterschiedlichen Frequenzen. Dabei wird auf die Genauigkeit des Sensors, die maximal messbare Entfernung und die Replizierbarkeit der Messwerte eingegangen. "Timing Budget" ist dabei die Zeit, die benötigt wird, um einen Messwert aufzunehmen. Die Frequenz lässt sich daraus mit dem Kehrbruch berechnen. "STDEV" steht für "standard deviation" oder auch Standardabweichung. Dieser Wert gibt die Streubreite der gemessenen Werten um den tatsächlichen Wert an. Je höher dieser Wert, desto geringer ist die Genauigkeit des Sensors.

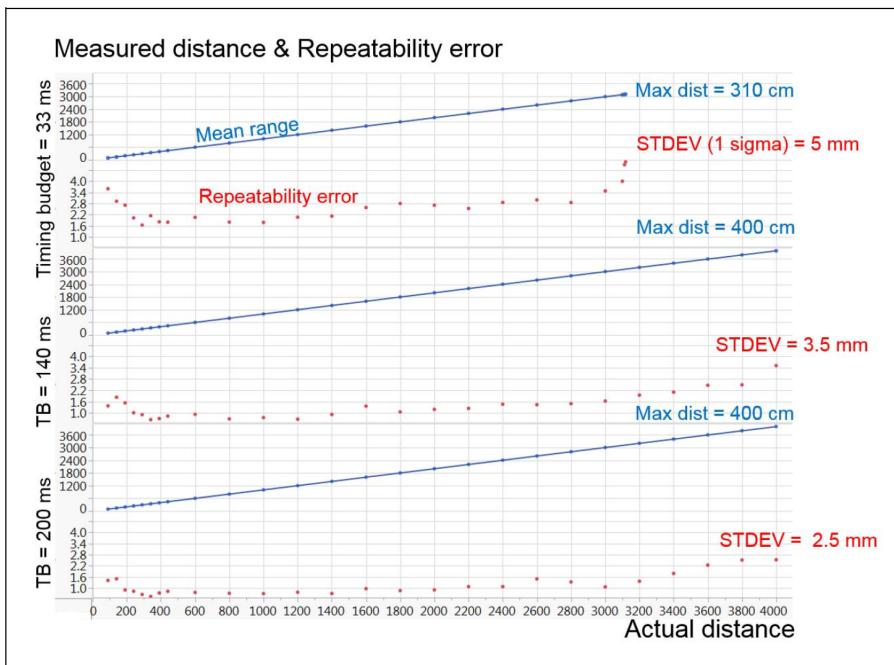


Abbildung 8.2: Abhängigkeit der Genauigkeit des VL53L1X Sensors von der Messfrequenz

Das obere Diagramm zeigt das Verhalten des Sensors bei einer Messfrequenz von 30,3 Hz. Bei dieser Frequenz kann unter perfekten äußereren Bedingungen nur eine maximale Distanz von 310 cm erreicht werden. Zudem ist die Genauigkeit des Sensors am schlechtesten. Mit kleiner werdender Messfrequenz nimmt die Genauigkeit zu und der maximal messbare Distanz beträgt bei entsprechenden Messverhältnissen 400 cm. Gute Messverhältnisse sind möglichst wenig Umgebungslicht und gut reflektierende Gegenstände. [21]

Der Sensor ist mit 4,9 x 2,5 x 1,56 mm sehr klein. Zudem müssen 12 Pins auf der Unterseite verlötet werden. Aufgrund mangelnder Ausrüstung zum Löten solchen Chips wird ein bereits verbauter Sensor auf einer Platine verwendet. Auf der Platine sind zudem benötigte Widerstände und Kondensatoren verbaut. Werden die 12 herausgeführten Pins der Platine verwendet, ist der Sensor um 90 Grad vertikal verdreht und kann nicht auf der dafür vorgesehenen Adapterplatte der Mechanik montiert werden. Deshalb wird eine weiter Adapterplatine entworfen, auf welche die Platine mit dem Sensor aufgesteckt werden kann und somit um 90 Grad gedreht wird.

– BILD, Schaltplan, Board...

Im Vergleich zum "TF Mini LIDAR" Sensor muss beim "VL53L1X" auf äußere Einflüsse wie Umgebungslicht oder Reflexionsverhalten der Oberflächen geachtet werden. Zudem ist die maximal messbare Entfernung laut Spezifikation zu gering. Da jedoch ein möglichst kostengünstiges System entwickelt werden soll, wird der Sensor trotzdem für Tests verwendet, da es der kostengünstigste Sensor auf dem Markt ist, welcher den Anforderungen annähern entspricht.

### 8.1.3 Tabellarischer Vergleich der Sensoren

Eine tabellarische Auflistung der relevanten Daten der beiden ausgewählten Sensoren vereinfacht den direkten Vergleich für spätere Auswertungen. Neben vor allem technischen Daten wird auch der Preis in Tabelle 8.2 aufgenommen.

Tabelle 8.2: Tabellarischer Vergleich TF Mini und VL53L1X

	<b>Tf Mini LIDAR</b>	<b>VL53L1X</b>
Preis [€]	35-40	10
Arbeitsbereich [cm]	30 - 1200	3-400
max. Messfrequenz [Hz]	100	50
Messungenauigkeit [%]	1-2	<1
Schnittstelle	UART	I <sup>2</sup> C

Es gilt zu beachten, dass sich beim Sensor VL53L1X einige Werte gegenseitig ausschließen. So ist bei einer Messfrequenz von 50 Hz beispielsweise keine Entfernung von 400 cm messbar. Diese gegenseitigen Einschränkungen sind in Kapitel 8.1.2 aufgeführt und erklärt.

## 8.2 Funktionseinheit Ausrichtung des Sensors

Der Sensor wird durch je einen Schrittmotor in der Horizontalen als auch Vertikalen bewegt. Die vertikale Ausrichtung erfolgt dabei direkt. Der Sensor ist über eine Adapterplatte direkt mit der Welle des Schrittmotors verbunden. Dadurch werden Bewegungen des Schrittmotors 1:1 auf den Sensor übertragen.

Die horizontale Ausrichtung erfolgt zusätzlich über einen Zahnriementrieb zur Kraftübertragung. Das Übersetzungsverhältnis entspricht 6:1.

### 8.2.1 Schrittmotoren

Das Drehen der Basis übernimmt ein bipolarer Hybrid-Schrittmotor der Bauform NEMA 17 und einem Vollschrittwinkel von 1.8°. Der Maximalstrom beträgt 1.2 A pro Phase bei einer Spannung von 4V.

Dieser Motor wurde gewählt, da er genug Drehmoment aufbringt, um die gesamte Basis drehen zu können. Das hohe Haltemoment von  $3,2 \frac{kg}{cm}$  verhindert ungewolltes Verdrehen der Basis während der Messung. Dadurch ist das System fehlerresistenter auf äußere Einflüsse. [17]

Zum Kippen des Lidar-Sensors wird ebenfalls ein bipolarer Hybrid-Schrittmotor verwendet. Dieser Motor befindet sich auf der sich drehenden Basis. Der Schwerpunkt des Motors befindet sich dabei unumgänglich einige Zentimeter neben der Drehachse. Deswegen sollte der Motor möglichst wenig Gewicht aufweisen, um die bei Drehung entstehende Unwucht so klein wie möglich zu halten. Zum vertikalen Kippen des Sensor wird nicht so viel Kraft benötigt, als für das Drehen der gesamten Basis. Auf Grund dessen reicht die Bauform Nema 11. Diese Bauform ist deutlich kleiner und dadurch leichter. Das Haltemoment reicht aus, um ungewolltes vertikales Verdrehen zu vermeiden.

## 8.2.2 Schrittmotortreiber

Zur Ansteuerung der Schrittmotoren wird der Schrittmotortreiber A4988 verwendet. Dieser ist bereits auf einer Trägerplatine mit Teilen der äußeren Beschaltung verbaut.

Der Motortreiber ermöglicht es, bipolar Schrittmotoren mit einer Motorspannung von 8 V - 35 V mit einem maximalen Phasenstrom von 2 A anzusteuern. [2]

Mit dem Motortreiber sind Mikroschritte realisierbar. Dabei sind halb, viertel, achtel und sechzehntel Schritte möglich. Der maximale Ausgangsstrom ist über einen Potentiometer auf der Trägerplatine stufenlos einstellbar.

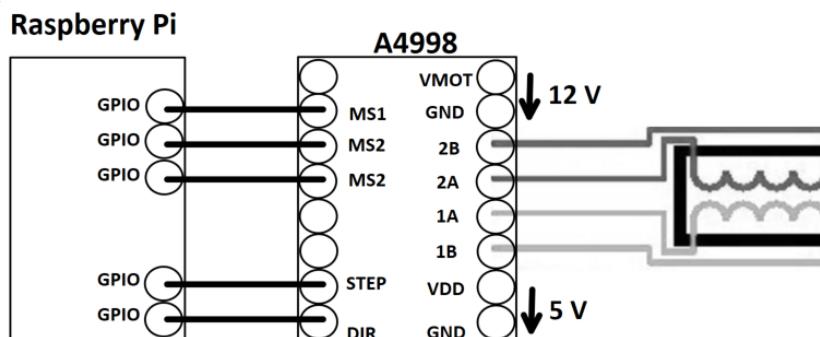


Abbildung 8.3: Schematische Beschaltung der A4988 Trägerplatine

Der Motor wird spulenweise an den Motortreiber angeschlossen. Dabei wird Pin 1A und 1B sowie 2A und 2B über jeweils eine Spule des Schrittmotors verbunden. Die Versorgungsspannung wird über ein 12 V Tischnetzteil bereitgestellt. Die Versorgungsspannung wird zusätzlich mit einem Stützkondensatoren geglättet. Der Kondensator hat eine Kapazität von 150  $\mu$ F. Als Kontrolleinheit für den Motortreiber dient ein Raspberry Pi. Die Logikspannungsversorgung des Treibers wird mit 5 V gespeist.

Die Anschlüsse "STEP", "DIR", sowie "MS1-MS3" werden mit GPIO's verbunden. Der Logikpegel an dem DIR-Pin legt die Bewegungsrichtung des Motors fest. Ein toggelndes

Tabelle 8.3: Schrittweite

MS1	MS2	MS3	Auflösung
Low	Low	Low	Vollschritt
High	Low	Low	Halbschritt
Low	High	Low	viertel Schritt
High	High	Low	achtel Schritt
High	High	High	sechzehntel Schritt

Signal am STEP-Pin führt zur Rotation des Motors. Pro steigende Flanke dreht sich der Motor um die mit MS1-MS3 eingestellte Schrittweite weiter.

MS1-MS3 dienen zum Einstellen der Schrittweite. Die Kombination der Logiklevel entscheidet dabei über die Schrittweite. Theoretisch wären somit  $2^3$  Kombinationen möglich. Es werden jedoch nur fünf Einstellungen benötigt. Die verfügbaren Kombinationen sind in Tabelle 8.3 aufgelistet. In der Tabelle ist der Zusammenhang von Schrittweite und Logiklevel an MS1-MS3 dargestellt.

Um Beschädigungen an den Schrittmotoren zu vermeiden, muss der maximale Strom durch die Spulen begrenzt werden. Dies kann man über einen Potentiometer auf der Oberseite der Trägerplatine machen. Dafür wird die Referenzspannung zwischen dem Potentiometer und Masse gemessen. Mit der Formel:

$$I_{max} = U_{Ref} \cdot 2 \quad (8.1)$$

$I_{max}$  = maximaler Strom pro Phase [A]

$U_{Ref}$  = Referenzspannung zwischen Potentiometer und Masse [V]

wird der maximale Strom bei gegebener Spannung berechnet. Durch verändern des Widerstandes des Potentiometer verändert sich die Referenzspannung und der Wert des maximalen Stroms ändert sich.

Unterschiedliche Bauweisen der Bauteile führen oftmals dazu, dass die Formel nur als grober Richtwert gewertet werden kann. Nach der groben Einstellung des Stromlimits mithilfe der Formel sollte der Strom bei aktivem Motor gemessen und gegebenenfalls noch angepasst werden.

Das Verändern der Schrittweiten hat ebenfalls einen Einfluss auf den maximalen Strom. Deshalb sollte der Treiber auf die Schrittweite eingestellt werden, bei der der maximale

Strom fließt.

Pro Motor wird ein Motortreiber verwendet.

## 8.3 Funktionseinheit Kalibrierung

Sowohl die horizontale als auch vertikale Ausgangsposition soll beim Starten einer Messung vom System selbstständig gefunden werden. Dabei wird der Sensor möglichst parallel zum Rahmen und dem Boden ausgerichtet. Dadurch wird spätere Nachbearbeitung der Daten hinsichtlich Ausrichtung überflüssig, was Zeit und Aufwand spart. Wird die Anfangskalibrierung der Achsen nicht oder nur ungenau vorgenommen, ist die spätere 3D Darstellung um einen bestimmten Winkel verdreht.

### 8.3.1 Lichtschranke

Zum automatischen Positionieren der Basis wird eine Infrarot Lichtschranke verwendet. Diese befindet sich unter der Zahnriemenscheibe und detektiert das Durchlaufen des daran befestigten Kalibrierzapfens. Lichtschranke und Zapfen sind so zueinander ausgerichtet, dass beim Detektieren des Zapfens der Sensor parallel zum Rahmen steht.

Auf dem Lichtschrankenmodul ist LM393 Komparator Integrated Circuit (IC) verbaut. Das Modul benötigt eine Versorgungsspannung von 5 V. Der Ausgangspin wird mit einem GPIO-Pin des Raspberry Pis verbunden. An diesem Ausgang liegt ein digitales Signal an, welches den Status der Lichtschranke darstellt. Liegt ein High-Signal an, befindet sich etwas zwischen der Lichtschranke und sie ist unterbrochen. Wechselt dieser Wert auf einen Low-Pegel, so ist sie nicht mehr unterbrochen [1].

-Bild-?

### 8.3.2 Gyroensor

Die Kalibrierung in vertikaler Richtung soll über eine Gyroskop realisiert werden. Dieses muss die absolute Ausrichtung in der vertikalen Richtung detektieren können.

Als Sensor wird der Beschleunigungssensor MPU 6050 verwendet. Dieser benötigt eine Versorgungsspannung von 3,3 V. Die Messwerte werden über I<sup>2</sup>C ausgegeben [13]. Sowohl Spannungsversorgung als auch Takt- und Datenleitung werden direkt mit dem Raspberry

Pi verbunden.

Der Sensor wird vor dem Befestigen an der Unterseite der Adapterplatte auf seine Funktion überprüft. Dabei wird festgestellt, dass der Sensor nicht die benötigte Genauigkeit liefert, um die genaue Positionierung des Sensors in vertikaler Richtung zu gewährleisten.

Die vertikale Ausrichtung erfolgt während der Tests manuell.

## 8.4 Platinen

Um einen Wechsel der Rechen- und Steuereinheit zu ermöglichen, wird eine Platine entworfen, die über ein 40adriges Flachbandkabel direkt mit allen Pins des Raspberry Pi's verbunden werden kann. Auf der Platine sind zudem Motortreiber und Spannungsversorgung verbaut. Alle benötigten Schnittstellen und zusätzliche GPIO's sind durch die Platine mit Stiftleisten verbunden.

Für die Schaltplan- und Layouterstellung wird die Software "EAGLE" von Autodesk verwendet.

### 8.4.1 Abwärtswandler

Einige Komponenten benötigen eine Spannung von 5V. Die Versorgungsspannung von 12 V muss somit verringert werden. Dazu wird ein Abwärtswandler des Typs MH-Mini-360 verwendet. Über ein Potentiometer auf der Oberseite des Moduls kann die Ausgangsspannung stufenlos eingestellt werden.

### 8.4.2 Schaltplan

Abbildung 8.4 zeigt die Beschaltung der Spannungsversorgung und Spannungswandlung. Über den Hohlstecker (links) wird die Platine mit 12 V versorgt. Der  $150 \mu\text{F}$  Kondensator dient zur Glättung der Eingangsspannung. Der Abwärtswandler wird mit den 12 V Eingangsspannung verbunden. Der Ausgang wird auf 5 V eingestellt.

Sowohl die 12 V als auch die 5 V können über Kurzschlussbrücken bzw. Schalter von der nachfolgenden Schaltung getrennt oder verbunden werden.

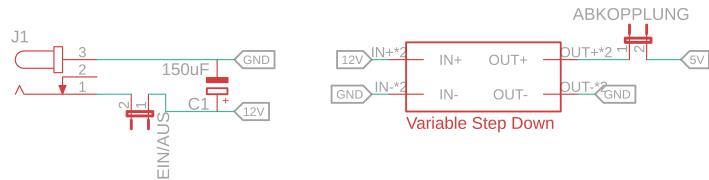


Abbildung 8.4: Schaltplan: Spannungsversorgung

Abbildung 8.5 zeigt die Beschaltung der Motortreiber wie in Kapitel 8.2.2 beschrieben. Die vier Anschlüsse für die Motorspulen werden mit Stifteleisten realisiert, an denen man den Motor später anstecken kann.

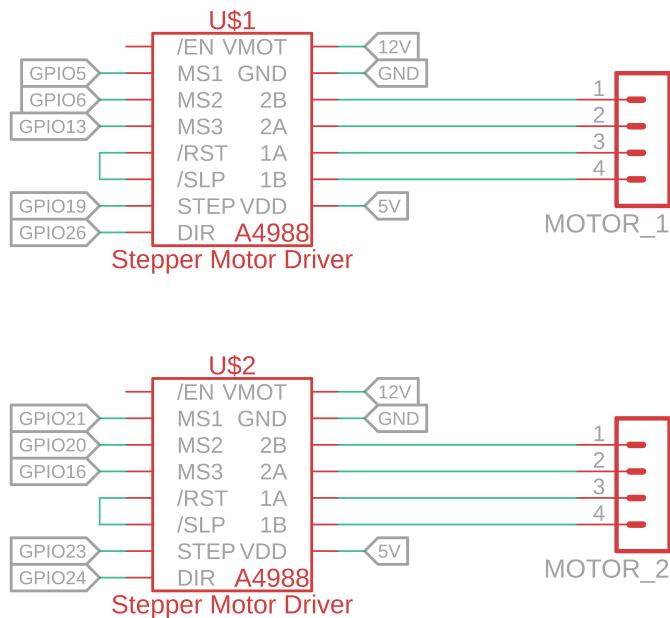


Abbildung 8.5: Schaltplan: Motortreiber

Abbildung 8.6 zeigt die Verbindung zum Raspberry Pi und die verschiedenen Schnittstellen. An den 40 Pins (links) kann ein Raspberry Pi über ein Flachbandkabel verbunden werden. Die benötigten Schnittstellen werden über Stifteleisten zugänglich gemacht. Es werden eine UART-, eine Serial Peripheral Interface (SPI)- und drei I<sup>2</sup>C- Schnittstelle verbunden. Es werden mehr Schnittstellen zugänglich gemacht, als benötigt. Dies garantiert Flexibilität für Weiterentwicklungen.

Ein Weiterer Anschluss ist mit den verschiedenen Spannungen der Platine verbunden.

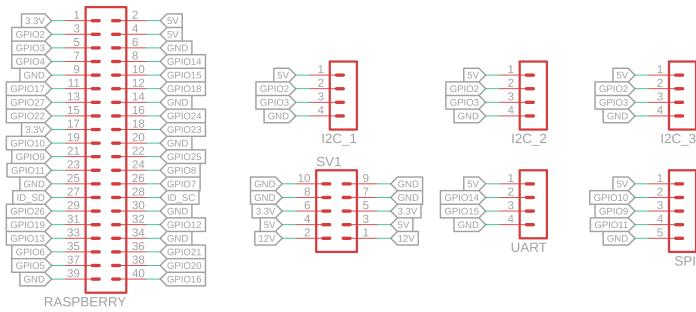


Abbildung 8.6: Schaltplan: Schnittstellen

Abbildung 8.7 zeigt die Schaltungen für den optionalen Anschluss von Status Led's und einem Luftkühler über Pins. Eine Statusled kann über einen strombegrenzenden Widerstand mit 5 V verbunden werden. Steht das System unter Spannung, leuchtet die Led. Eine weitere Led kann mit dem Raspberry Pi über einen Transistor angesteuert werden. Die Funktion der Led kann im Code direkt definiert werden.

An zwei weiteren Pins kann ein Luftkühler an 12 V angeschlossen werden. Die Steuerung des Kühlers ist über einen Transistor mit dem Raspberry Pi möglich.

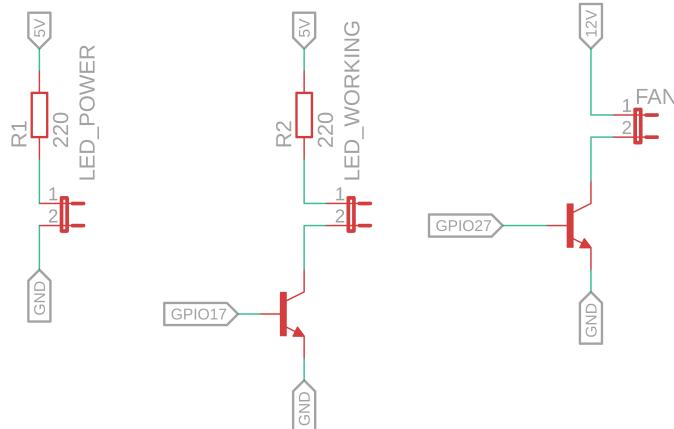


Abbildung 8.7: Schaltplan: Leds und Kühler

### 8.4.3 Layout

Die in Kapitel 8.4.2 gezeigten Schaltpläne werden in "EAGLE" zu einem Layout verarbeitet. Dabei wird vor allem auf eine praktische Anordnung der Komponenten und sinnvolle Leiterbahnführung geachtet.

Die Leiterbahnen für die 12 V Spannungsversorgung haben eine Breite von 1,78 mm. Die 5 V Leiterbahnen eine Breite von 1,27 mm. Für restlichen Leiterbahnen reicht eine Breite

von 0,82 mm.

Die roten Verbindungen in Abbildung 8.8 sind Verbindungen, die nach der Fertigung der Platine mit Drahtbrücken hergestellt werden müssen.

Die Platine hat eine Größe von 7,5 mm auf 10 mm und wird gefräst. Um die Übersichtlichkeit über die vielen Anschlussmöglichkeiten zu behalten, werden die jeweiligen Anschlüsse beschriftet.

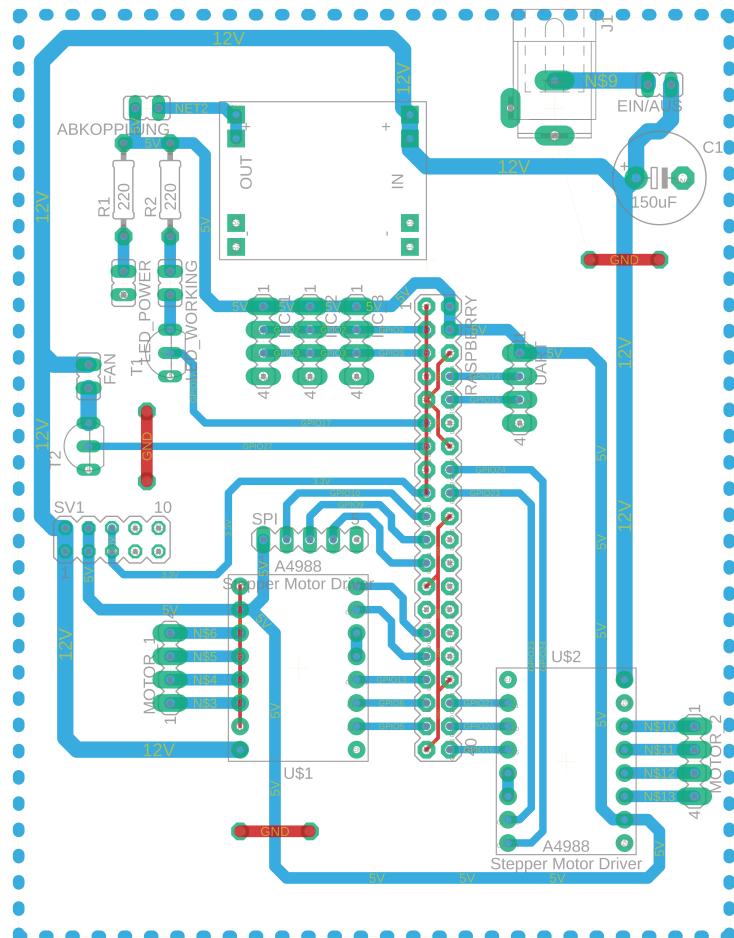


Abbildung 8.8: Platinenlayout

# 9 Code

Die gewählte Sprache in welcher die Steuerung realisiert ist, ist Python. Python wurde gewählt, da mittels dieser die GPIOs des Raspberry Pi sehr einfach mittels einer Bibliothek ansteuerbar sind. Zudem ist Python eine sehr schnelle und weit verbreitete hochentwickelte Programmiersprache.

Bei der Erstellung des Codes, welcher das System steuert wurde von Anfang an eine Objektorientierte Vorgehensweise gewählt, um eine möglichst Reibungslose und fortschrittliche Umsetzung zu realisieren.

Der gesamte Code wurde auf drei Dateien aufgeteilt, dies dient zum einen zur besseren Übersichtlichkeit, zum anderen erhielt jede Klasse eine eigene Datei.

## 9.1 Motor

Die erste Datei und Klasse beschäftigt sich mit der Ansteuerung der Schrittmotoren. Sie benötigt zwei extra Bibliotheken (Listing 9.1). Die 'time' Bibliothek wird benötigt, um zwischen verschiedenen Befehlen 'schlafen' zu können, sprich das Programm pausieren zu können. Die 'RPi.GPIO' Bibliothek wird benötigt um die GPIOs des Raspberry PI ansteuern zu können.

```
1 import time
2 import RPi.GPIO as GPIO
```

Listing 9.1: Bibliotheken der Motor Klasse

### 9.1.1 Konstruktor

Der Konstruktor der Klasse beschäftigt sich mit der Deklaration von Variablen und dem zuweisen der dem Konstruktor übergebenen Parameter.

Im Falle der Motor Klasse bekommt der Konstruktor sechs Übergabeparameter, wovon

allerdings ein Parameter ('self') eine Referenz auf das eigene Objekt ist.

Die restlichen übergebenen Parameter sind die GPIOs, welche für die Ansteuerung des Motortreibers benötigt werden.

Bei einem Blick auf den Code des Konstruktors (Listing 9.2) sieht man die Übernahme der Übergabeparameter in Klasseneigene Variablen (Zeile 2-6). Anschließend wird die Kommunikationsrichtung der GPIOs festgelegt (Zeile 7 - 11). In diesem Fall werden alle Pins als Ausgang benötigt.

Außerdem wird den GPIOs direkt ein Zustand zugewiesen (Zeile 12 - 16), in diesem Fall ist die Konfiguration so, dass der Motor Treiber mit Achtelschritten arbeitet und den Motor gegen den Uhrzeigersinn drehen lässt.

```
1 def __init__(self, Step, Dir, MS1, MS2, MS3):
2     self.step = Step
3     self.dir = Dir
4     self.MS1 = MS1
5     self.MS2 = MS2
6     self.MS3 = MS3
7     GPIO.setup(self.step, GPIO.OUT)
8     GPIO.setup(self.dir, GPIO.OUT)
9     GPIO.setup(self.MS1, GPIO.OUT)
10    GPIO.setup(self.MS2, GPIO.OUT)
11    GPIO.setup(self.MS3, GPIO.OUT)
12    GPIO.output(self.step, GPIO.LOW)
13    GPIO.output(self.dir, GPIO.LOW)
14    GPIO.output(self.MS1, GPIO.HIGH)
15    GPIO.output(self.MS2, GPIO.HIGH)
16    GPIO.output(self.MS3, GPIO.LOW)
```

Listing 9.2: Konstruktor der Motor Klasse

### 9.1.2 Bewegen des Motors

Die Motor Klasse besitzt zudem noch eine Funktion, mittels welcher sich der jeweilige Motor bewegen lässt (Listing 9.3). In der Funktion wird zunächst die Drehrichtung je nach Übergabeparameter gesetzt (Zeile 2 - 5), und anschließend ein bzw. je nachdem wie viele Schritte gefordert werden ausgeführt. Um einen kompletten Schritt zu vollenden, wird der dafür vorgesehene Pin des Motortreibers Ein und wieder Aus geschaltet. Die Zeit zwischen diesen beiden Vorgängen kann über einen Übergabeparameter der Funktion eingestellt werden (Zeile 8 - 13). Dies bestimmt direkt die Drehgeschwindigkeit des Motors. Wobei allerdings eine kleinere übergebene Zeit eine schnellere Drehung des Motors produziert.

```
1 def moveMotor(self, dir, step, speed):
2     if(dir):
3         GPIO.output(self.dir, GPIO.HIGH)
4     else:
5         GPIO.output(self.dir, GPIO.LOW)
6
7     i = 0
8     while i < step:
9         GPIO.output(self.step, GPIO.HIGH)
10        time.sleep(speed)
11        GPIO.output(self.step, GPIO.LOW)
12        time.sleep(speed)
13     i += 1
```

Listing 9.3: Funktion zum Bewegen des Motors

## 9.2 Lidar

Auch der LIDAR Sensor hat eine eigene Datei sowie Klasse bekommen, dies soll dazu dienen, um mehrere verschiedene Sensoren konfigurieren zu können und diese dann schnell und einfach mit denselben Funktionen auswählen zu können.

Die Klasse ist in ihrer jetzigen Form bereits in der Lage zwei Verschiedene LIDAR Sensoren zu bedienen. Die Lidar Klasse benötigt zwei Bibliotheken (Listing 9.4), mit der ersten kann eine serielle Verbindung erstellt werden. Die Zweite Bibliothek wird benötigt, um einen der zwei Möglichen LIDAR Sensoren anzusteuern *TODO: Referenz VL53L1X*.

```
1 import serial
2 import VL53L1X
```

Listing 9.4: Bibliotheken der Lidar Klasse

### 9.2.1 Konstruktor und Variablen

Die LIDAR Klasse besitzt zwei Variablen. Die Variable "dist" wird verwendet, um die gemessene Entfernung zu speichern und auf diese Zugreifen zu können.

Die zweite Variable wird als Flag bei Verwendung des LIDAR Sensors 'TFMini' (Kapitel 8.1.1) benötigt (Listing 9.5).

Der Konstruktor der Klasse ist zudem in der Lage je nachdem, welche Parameter angegeben

werden, die korrekte Verbindung herzustellen. Je nachdem welche Werte angegeben und welche als "None" definiert werden, stellt der Konstruktor entweder eine Verbindung über UART (Zeile 9) oder I<sup>2</sup>C (Zeile 11 - 13) her.

```

1 class LIDAR():
2     dist = 0
3     recievedData = False
4
5     def __init__(self, uart, i2c):
6         self.uart = uart
7         self.i2c = i2c
8         if(self.uart != None and self.i2c == None):
9             self.ser = serial.Serial(self.uart, 115200, timeout=1)
10        else:
11            self.tof = VL53L1X.VL53L1X(i2c_bus=1, i2c_address=i2c)
12            self.tof.open()
13            self.tof.start_ranging(3)

```

Listing 9.5: Konstruktor der Lidar Klasse

## 9.2.2 Aufnehmen von Messdaten

Die Funktion um anschließend Daten vom LIDAR Sensor zu bekommen ist auch in der Klasse definiert, somit kann für egal welchen Sensortyp über die selben Funktionsaufrufe die Distanz ermittelt werden.

```

1     def getData(self):
2         if(self.uart != None and self.i2c == None):
3             self.ser.reset_input_buffer()
4             while(self.recievedData != True):
5                 while(self.ser.in_waiting <= 9):
6                     if((b'Y' == self.ser.read()) and (b'Y' == self.ser.
7                         read())):
7                         Dist_L = self.ser.read()
8                         Dist_H = self.ser.read()
9                         self.dist = (ord(Dist_H) * 256) + (ord(Dist_L))
10                        for i in range (0,5):
11                            self.ser.read()
12                        self.recievedData = True
13                        break
14                    else:
15                        self.dist = self.tof.get_distance() # Entfernung in mm

```

16

```
self.dist = self.dist/10.0
```

Listing 9.6: Funktion um Distanz vom LIDAR Sensor zu erhalten

In Listing 9.6 kann man sehen, dass ähnlich wie im Konstruktor je nachdem welcher Sensor 'ausgewählt' wurde unterschiedliche Methoden verwendet werden um Daten zu bekommen. Der erste Abschnitt in Zeile 3 - 13 ist für die Verwendung eines Sensors mittels UART gedacht. Da UART ein Serieller Bus ist, auf welchen vom Slave konstant Daten geschickt werden, wartet diese Funktion so lange, bis neue Daten ankommen. Die neuen Daten werden durch zwei aufeinander folgende 'Y' gekennzeichnet. Anschließend werden die Zwei bit für die Entfernung gespeichert (Zeile 7 & 8) und zur Gesamtdistanz zusammengefügt (Zeile 9). Anschließend wird die bereits erwähnte Flag der Klasse gesetzt, damit nur ein einzelner Wert aufgenommen wird.

Die Zweite Methode in Zeile 15 - 16 ist deutlich einfacher, da hierbei eine Bibliothek verwendet werden kann und die Distanz lediglich in die richtige Größe konvertiert werden muss (Zeile 15 - 16).

## 9.3 Steuerung

Die dritte und letzte Datei beschäftigt sich mit der generellen Steuerung des Systems und dem Initialisieren und Aufrufen der Klassen und ihrer Funktionen.

Für die Steuerung des Systems werden einige Bibliotheken mehr benötigt.

```
1 # Bibliotheken
2 import time
3 import datetime
4 import math
5 import RPi.GPIO as GPIO
6
7 # Eigene Dateien
8 import Lidar
9 import Motor
10
11 # GPIO Nummerierung gleich der Pin Nummer
12 GPIO.setmode(GPIO.BOARD)
13 GPIO.setwarnings(False)
```

Listing 9.7: Bibliotheken zur Steuerung des Systems

Die Bibliotheken in Zeile 2 & 3 (Listing 9.7) werden für die Benennung der Dateien, welche produziert werden benötigt. Die 'math' Bibliothek wird für einige Berechnungen benötigt und die 'RPi.GPIO' wird wie bereits erwähnt benötigt und die GPIOs des Raspberry Pi möglichst einfach anzusteuern. Anschließend werden dann noch die zwei Klassen importiert welche in den vorangegangenen Abschnitten erklärt wurden. Zudem wird noch der Modus der GPIO Nummerierung festgelegt. In diesem Fall ist der Modus gleich der Nummerierung der Pins auf dem Board. *TODO: Bild GPIO pinout*

### 9.3.1 Pin Definitionen und Initialisieren der Klassen

Da wie in den Vorangegangenen Kapiteln erläutert wurde Klassen für Motor und Lidar erstellt wurden müssen diese nun auch aufgerufen und initialisiert werden. Zudem sind weitere GPIOs nötig um das gesamte System zu steuern.

```

1 # Pins & Definitionen
2 workingLED = 11
3 fan = 13
4 lightGate = 23 #SPI SCLK --> In Version 2 der Platine eigenen Pin
                 zuweisen
5
6 # Motor 1, Nema 11
7 M1 = Motor.MOTOR(31,29,37,35,33)
8
9 # Motor 2, Nema 17
10 M2 = Motor.MOTOR(18,16,36,38,40)
11
12 # LIDAR Sensor
13 lidar = Lidar.LIDAR('/dev/ttyAMA0', none)
14 #lidar = Lidar.LIDAR(None, 0x29)
```

Listing 9.8: Initialisieren von Variablen und Klassen

Zunächst werden die Pins für die verschiedenen auf der Platine vorgesehenen Funktionen definiert (Listing 9.8). Eine Anmerkung hierzu ist, dass auf der Platine versäumt wurde einen Pin für die Lichtschranke zur Positionierung bereitzustellen, daher wurde hier der Pin verwendet, welcher eigentlich für den seriellen Takt des SPI zuständig ist. Außerdem wurden Pins für eine Status LED und einen Lüfter bereitgestellt.

Nach den normalen Pin Deklarationen werden die beiden Motoren durch die Klassen initialisiert. Dazu werden wie im Kapitel der Motorklasse beschrieben die verschiedenen Pins

zur Ansteuerung des Motortreibers dem Konstruktor der Klasse übergeben. Anschließend kann der Motor mittels den in der Klasse definierten Funktionen gesteuert werden. Zuletzt muss nur noch der LIDAR Sensor initialisiert werden, dazu kann wie in Zeile 13 & 14 zu sehen ist eine der beiden Initialisierungsmöglichkeiten gewählt werden, um entweder einen Sensor mittels UART oder I<sup>2</sup>C zu verwenden.

### 9.3.2 Zusätzliche Funktionen

Nachdem alle benötigten Variablen für die GPIOs definiert sind, werden noch einige Funktionen benötigt um einen schöneren und übersichtlicheren Code zu erzeugen (Listing 9.9).

```
1 # Funktion um GPIO's zu Initialisieren
2 def initGPIO():
3     GPIO.setup(workingLED, GPIO.OUT)
4     GPIO.output(workingLED, GPIO.LOW)
5     GPIO.setup(fan, GPIO.OUT)
6     GPIO.output(fan, GPIO.LOW)
7     GPIO.setup(lightGate, GPIO.IN)
8
9 def homeAxis():
10    while(GPIO.input(lightGate)!=GPIO.HIGH):
11        M2.moveMotor(1,1,0.001)
12    count = 0
13    while(GPIO.input(lightGate)==GPIO.HIGH):
14        M2.moveMotor(1,1,0.001)
15        count += 1
16    M2.moveMotor(1,36,0.001)
```

Listing 9.9: Funktionen für die Übersichtlichkeit des Codes

Die erste Funktion (Zeile 2 - 7) dient dazu, um die übrigen GPIOs zu initialisieren und diesen einen Startwert zu geben. Hier werden die GPIOs welche für die LED und den Lüfter vorgesehen sind als Ausgang deklariert und als 'LOW' initialisiert. Der GPIO Pin welcher für die Lichtschranke vorgesehen ist wird als Eingang deklariert.

Die zweite Funktion wird benötigt, um das System in horizontaler Richtung in die Ausgangslage zu bringen. Dazu wird der Motor 2, welcher für den Polarwinkel zuständig ist, so lange gedreht, bis dieser die Lichtschranke erreicht, und diese wieder verlässt. Da die Lichtschranke nicht zu 100% am Kreisscheitelpunkt positioniert ist, wird nach verlassen

der Lichtschranke mit einem Manuellen Kalibrationswert (Zeile 16) der Polarwinkel in Nulllage gebracht.

### 9.3.3 Variablen deklaration und aufrufen von Funktionen

Bevor mit dem eigentlichen Messen begonnen werden kann müssen noch einige Variablen definiert und Funktionen aufgerufen werden.

```
1 initGPIO()
2 lidar.getData()
3 lidar.recievedData = False
4 homeAxis()
5 time.sleep(2)
6
7 stepsFullRotM2 = 200*8*6
8 stepsQuartRotM1 = 50*4
9
10 M1dir = False
11 M2dir = True
12
13 lidar.getData()
14 print(lidar.dist)
15 lidar.recievedData = False
16
17 data = open("data_"+datetime.datetime.fromtimestamp(time.time()).
18     strftime('%Y-%m-%d_%H-%M-%S')+".csv", "w")
18 data.write("Nr;Distance;Azimuth;Elevation\n")
19 valueNr = 1
```

Listing 9.10: Aufrufen von Funktionen und Variablen deklaration

In Zeile 1 (Listing 9.10) werden zuerst die GPIOs initialisiert. Anschließend wird ein Wert vom LIDAR ausgelesen und die Flag des LIDARs wieder auf 'False' gesetzt, damit dieser anschließend wieder Messwerte aufnehmen kann. Dies dient zum Funktionstest, bevor das System bewegt wurde. Wenn der Sensor beispielsweise nicht korrekt verbunden wurde würde das System abbrechen bevor etwas Bewegt wurde. Danach wird das System in die Ausgangslage positioniert und eine kurze Zeit gewartet, bevor es mit dem eigentlichen Messvorgang losgeht.

Wie in Zeile 7 & 8 zu sehen werden anschließend noch die Anzahl der benötigten Schritte für eine ganze Umdrehung und eine Viertelumdrehung berechnet. Diese Werte werden

benötigt, um die Grenzen für die Schleifen der Messung festzulegen. Ein Schrittmotor macht pro Umdrehung 200 ganze Schritte, da eine möglichst hohe Auflösung gewünscht ist, wird die größtmögliche Mikroschritt Zahl der Motortreiber verwendet. Im Fall der verwendeten Motortreiber sind das Achtelschritte, allerdings hat der Motortreiber 1, welcher für die Vertikale Achse zuständig ist einen Fehler, wodurch nur viertel Schritte möglich sind. In Zeile 10 & 11 werden die Startrichtungen für die Motoren festgelegt. Anschließend wird ein Messwert in die Konsole ausgegeben, um dem Bediener eine Rückmeldung zu geben, dass das System ordnungsgemäß funktioniert.

### 9.3.4 Erstellen der Datei zum Speichern der Daten

Ein letzter Schritt ist noch nötig, bevor mit der Messung begonnen werden kann (Listing 9.11). Dabei wird die Datei in welcher die Messwerte abgespeichert werden erstellt. Die Datei wird dabei immer mit "data" und dem Aktuellen Datum sowie Timestamp versehen. Als letzte Aktion vor der Messung wird der Index des ersten Messwertes gleich 1 gesetzt.

```

1 data = open("data_" + datetime.datetime.fromtimestamp(time.time()).strftime('%Y-%m-%d_%H-%M-%S') + ".csv", "w")
2 data.write("Nr;Distance;Azimuth;Elevation\n")
3 valueNr = 1

```

Listing 9.11: Erstellen der Datei zum Speichern der Daten

Die comma separated values (csv) Datei in welcher die Daten gespeichert werden ist nach folgendem Schema aufgebaut:

Tabelle 9.1: Beispieldatei

Nr	Distance	Azimuth	Elevation
1	105	0	0
2	105	0	0.18
3	106	0	0.36

Jeder Wert welcher in die Datei gespeichert wird bekommt einen Index, somit kann man einen Überblick behalten, wie viele Messwerte aufgezeichnet wurden. Die Zweite Spalte,

welche in der Datei steht, ist für die Distanz vorgesehen, welche vom LIDAR Sensor ermittelt wird. Die beiden letzten Spalten der dienen der Speicherung des Winkels welcher aus der Position der Schrittmotoren ermittelt wird. Die Distanz wird in *cm* gespeichert und die beiden Winkel in Grad.

### 9.3.5 Schleifen der Steuerung

```

1 countM1 = 0
2 countM2 = 0
3 while(countM1 < stepsQuartRotM1):
4     if(M2dir):
5         while(countM2 < stepsFullRotM2):
6             lidar.getData()
7             lidar.recievedData = False
8             data.write(str(valueNr) + ";" + str(lidar.dist) + ";" + str
9                 (360.0*countM2/stepsFullRotM2) + ";" + str(90.0*countM1/
10                stepsQuartRotM1) + "\n")
11             M2.moveMotor(M2dir,4,0.00025)
12             valueNr += 1
13             countM2 += 4
14     else:
15         while(countM2 >= 0):
16             lidar.getData()
17             lidar.recievedData = False
18             data.write(str(valueNr) + ";" + str(lidar.dist) + ";" + str
19                 (360.0*countM2/stepsFullRotM2) + ";" + str(90.0*countM1/
20                stepsQuartRotM1) + "\n")
21             M2.moveMotor(M2dir,4,0.00025)
22             valueNr += 1
23             countM2 -= 4
24             M1.moveMotor(M1dir,4,0.0005)
25             countM1 += 4
26             M2dir = not M2dir
27
28 data.stop()
29 lidar.tof.stop_ranging()
```

Listing 9.12: Messen und Aufzeichnen der Entfernung

Da in Python keine konventionellen 'for' Schleifen möglich sind, müssen 'while' Schleifen verwendet werden. Dazu wird eine extra Variable benötigt (Zeile 1 & 2 Listing 9.12), um zu Überwachen wie oft die Schleife schon durchlaufen wurde. Da in diesem Fall zwei 'while'

Schleifen ineinander verschachtelt wurden, werden auch zwei Variablen benötigt um die Anzahl der Durchläufe zu überwachen.

Die äußere Schleife wird nur so oft durchlaufen, bis der Motor 1, welcher für die Neigung des Polarwinkles zuständig ist,  $90^\circ$  erreicht. Im inneren dieser Schleife gibt es eine Besonderheit. Da sich die Kabel des oberen Aufbaus nur endlich oft drehen können bis diese verschließen, muss das System nach jeden  $360^\circ$  die Richtung umkehren. Dafür werden 2 verschiedene Schleifen benötigt, da später der Winkel anhand des Counters der Schleifendurchläufe berechnet wird. Prinzipiell passiert in den Schleifen in Zeile 5 - 11 und Zeile 13 - 19 aber exakt das selbe. Zuerst werden neue Daten vom LIDAR Sensor angefragt, und die Flag des Sensors gesetzt. Anschließend wird der ermittelte Wert zusammen mit den aktuellen Winkeln und einem Index in die '.csv' Datei gespeichert. Die Winkel werden dabei in Grad angegeben (Zeile 8 & 16).

Nachdem die Werte gespeichert wurden, wird der Motor bewegt, in diesem Fall um vier Schritte, was auch zeigt, dass das System in diesem Codebeispiel nur mit einem Viertel der Auflösung in Horizontaler Richtung arbeitet. Anschließend wird der Index sowie der Schleifenzähler erhöht. Beim Schleifenzähler besteht der große Unterschied zwischen den beiden 'while' Schleifen, da einmal die Schleife von 'vorne' und einmal von 'hinten' durchlaufen wird. Die Schrittweite vier ergibt sich aus der Anzahl der Schritte welche auch dem Motor übergeben wurden.

Wenn die innere Schleife eine volle  $360^\circ$  Drehung also eine komplette Rotation durchlaufen hat, wird der Motor 1, welcher für den Polarwinkel zuständig ist ebenfalls bewegt, in diesem Fall auch um vier Schritte was ein Viertel der Vertikalen Auflösung bedeutet. Ebenfalls der zweite Schleifenzähler wird um vier erhöht, und die Richtung des Motors für die Horizontale Drehung wird umgekehrt, bevor die innere Schleife aufs neue durchlaufen wird.

Wenn die gesamte Messung abgeschlossen ist wird die Datei welche die Daten enthält geschlossen und die Messung gestoppt. Die Datei kann nun weiterverwendet und ausgewertet werden.

# 10 Auswertung und Darstellung mit Matlab

Die Daten werden nach Beendigung eines Scans exportiert und auf einem separaten PC ausgewertet. Die Auswertung und Darstellung erfolgt mit Matlab. Matlab bietet sich an, da große Datenmengen schnell ausgewertet und dargestellt werden können. Zudem sind bereits Kenntnisse zum Importieren und Darstellen von .csv Dateien vorhanden.

Die .csv Datei enthält die Rohdaten. Die Rohdaten bestehen pro Datenwert aus der Entfernung vom Messpunkt bis zum Hindernis in Metern. Zudem ist jedem Entfernungswert der Azimuth und die Elevation im Bezug zum jeweils gesetzten Nullpunkt zugeordnet. Die Messwerte sind fortlaufend nummeriert. Diese Werte müssen zuerst aus der .csv Datei in Matlab importiert werden. Anschließend erfolgt die Aufteilung des Datensatzes in die Vektoren Entfernung, Azimuth und Elevation. Die beiden Winkelwerte zusammen mit dem Entfernungswert stellen Kugelkoordinaten dar. Diese müssen zur Darstellung in Matlab in kartesische Koordinaten umgewandelt werden.

## 10.1 Importieren und Zuordnen der Messwerte

Im ersten Teil des Auswertungs- und Darstellungsprogramms wird die .csv-Datei als gesamtes in Matlab importiert. Anschließend werden die einzelnen Spalten dementsprechenden Variablen zugeordnet, um die spätere Auswertung zu erleichtern.

Das Importieren der Daten erfolgt über die "importdata" Funktion von Matlab, die es ermöglicht, Datensätze aus einer separaten Datei zu lesen. Das erste Argument der Funktion ist der relative Dateipfad zu der einzulesenden Datei. Dieser wird in Zeile 5 festgelegt. Das zweite Argument steht für das Trennzeichen, mit dem einzelne Elemente in Daten abgetrennt werden. Bei .csv-Dateien ist dies ein Semikolon. Der letzte Übergabeparameter gibt an, wie viele Kopfzeilen importiert werden sollen.

Die Importierte Daten liegen anschließend als Struct in der Variable "data" vor. Diese Struct enthält zum einen die Kopfzeile als Feld und zum andern die Messwerte ohne Kopfzeile. Für die weitere Verwendung der Daten, werden nur die Messwerte benötigt. Diese werden mit Zeile 8 extrahiert.

Anschließend werden in Zeile 10 bis 12 die einzelnen Spalten separiert und einer passenden Variablen zugeordnet.

```
1 % Anwendung zur Darstellung einer 3D Punktewolke aus einem LIDAR System
2 clear all;
3
4 % Importieren und Zuordnen der Messwerte
5 file = 'Messwerte-05-02/Aufloesung-hoch.csv';
6
7 data = importdata(file, ';' ,1);
8 data = data.data;
9
10 distance = data(:,2);
11 azimut = data(:,3);
12 elevation = data(:,4)
```

Listing 10.1: Importieren und Zuordnen von .csv Dateien

## 10.2 Umwandlung von Kugelkoordinaten zu kartesischen Koordinaten

Die Messpunkte liegen hardwarebedingt als Kugelkoordinaten vor. Dies bedeutet, dass jeder Punkt aus dem Abstand  $r$  zum Zentrum O, dem Polarwinkel  $\theta$  und dem Azimutwinkel  $\phi$  definiert wird.

Der Abstand  $r$  wird durch die Distanz des Punktes P von 0 bestimmt. Der Polarwinkel  $\theta$  ist der Winkel zwischen Flächennormalen und dem Vektor OP. Das Gegenstück dazu ist die Höhe. Der Polarwinkel reicht von 0 bis  $\pi$ .

Der Azimutwinkel ist der Winkel zwischen der x-Achse und der Projektion der Strecke OP auf die xy Ebene. Dieser Winkel reicht je nach Definition von  $-\pi$  bis  $\pi$  oder von 0 bis  $2\pi$ . Für das Lidar System wird die zweite Definition verwendet.

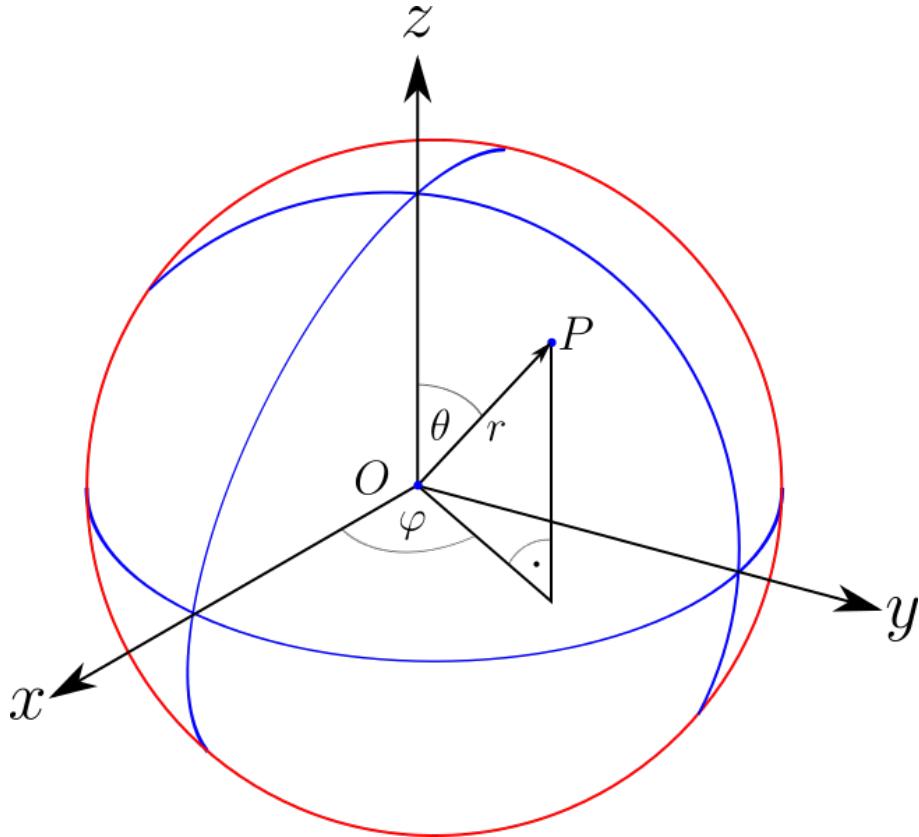


Abbildung 10.1: Kugelkoordinaten

Da dass das Darstellen von Kugelkoordinaten in Matlab nicht ohne weiteres möglich ist, werden die Messwerte in kartesische Koordinaten umgerechnet. Die Umrechnung erfolgt mit Formel 10.1:

$$\begin{aligned} x &= r \cdot \cos(\theta) \cdot \cos(\phi) \\ y &= r \cdot \cos(\theta) \cdot \sin(\phi) \\ z &= r \cdot \sin(\theta) \end{aligned} \quad (10.1)$$

$r$  = Abstand des Punktes zum Zentrum [m]

$\theta$  = Polarwinkel [ $^\circ$ ]

$\phi$  = Azimutwinkel [ $^\circ$ ]

In Matlab wird jeder einzelne Punkt innerhalb einer For-Schleife mit der oben genannten Formel umgewandelt und die drei Koordinaten  $x, y$  und  $z$  für kartesische Koordinaten

gespeichert. Als Endwert der Schleife wird die Zeilenanzahl des Datensatzes verwendet. Somit ist dieser Wert variabel und muss nicht für jeden Datensatz spezifisch angepasst werden.

Die Berechnung befindet sich zudem in einer If-Abfrage, welche dazu dient, offensichtliche Messfehler zu entfernen. Alle Koordinaten, bei denen die Entfernung höher als 10 Meter ist, werden gelöscht. Die 10 Meter wurden auf experimenteller Basis und aufgrund der Größe des vorher definierten Standardraums festgelegt. Beim Lidar TF Mini werden nicht messbare Entfernungen mit einer Entfernung von 34999 Metern angegeben. Diese werden somit mit dieser Abfrage ebenfalls gefiltert.

Matlab rechnet bei trigonometrischer Funktionen mit dem Radian. Die Winkel des Lidar Systems sind in Grad angegeben, weshalb sie innerhalb der Berechnung mit der Funktion deg2rad() in Radian umgerechnet werden müssen.

```

1 for i = 1:length(data)
2   if(distance(i) < 1000)
3     x(i) = -distance(i)*cos(deg2rad(elevation(i)))*cos(deg2rad(azimuth(i)))
4       );
5     y(i) = distance(i)*cos(deg2rad(elevation(i)))*sin(deg2rad(azimuth(i)))
6       );
7     z(i) = distance(i)*sin(deg2rad(elevation(i)));
8   else
9     end
10 end

```

Listing 10.2: Umwandlung von Kugelkoordinaten zu kartesischen Koordinaten

## 10.3 Darstellung der Messwerte

Im letzten Teil des Programms werden die kartesischen Koordinatenpunkte in eine 3D-Darstellung umgewandelt. Zudem wird die Skalierung der Achsen festgelegt.

```

1 %plot3(x,y,z)      %Darstellung mit Linien
2 %plot3(x,y,z, '*)    %Darstellung mit Asteriskus
3 plot3(x,y,z, '.')    %Darstellung mit kleinen Punkten

```

Listing 10.3: Darstellung der Messwerte

Die Darstellung der kartesischen Koordinaten erfolgt über die "plot3()" Funktion. Diese Funktion ermöglicht es, rotierbare, dreidimensionale Darstellungen anzufertigen. Die ersten

drei Argumente der Funktion sind die Vektoren mit den jeweiligen Koordinaten. Mit dem vierten Argument kann man die Darstellungsart der einzelnen Punkte festlegen. Anhand eines Datensatzes werden drei verschiedene Möglichkeiten auf Vor- und Nachteile überprüft.

Übergibt man der Funktion keinen Parameter, werden die Punkte durch schmale Linien verbunden. Dadurch entsteht ein dreidimensionaler Raum, mit sehr feiner Darstellung. Konturen sind dabei sehr gut zu erkennen. Zudem kann man den Verlauf der Messwertaufnahme erkennen.

Nachteil dieser Darstellungsart ist, dass auch Messfehler verbunden werden, wodurch es zu fehlerhaften Darstellungen kommt. Dies ist beispielsweise in Abbildung 10.2 zu erkennen.

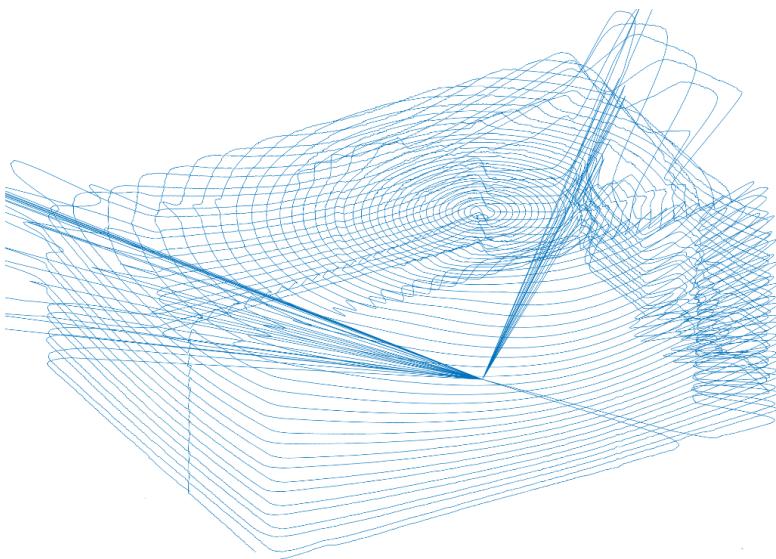


Abbildung 10.2: Darstellung mit Linien

Einen weitere Möglichkeit der Darstellung sind Asterisken. Diese sind relativ zu den Linien sehr groß. Räume werden auch mit weniger Messpunkten erkennbar. Dadurch verschwimmen jedoch Aufnahmen mit höherer Auflösung und Details sind nicht mehr so gut erkennbar.

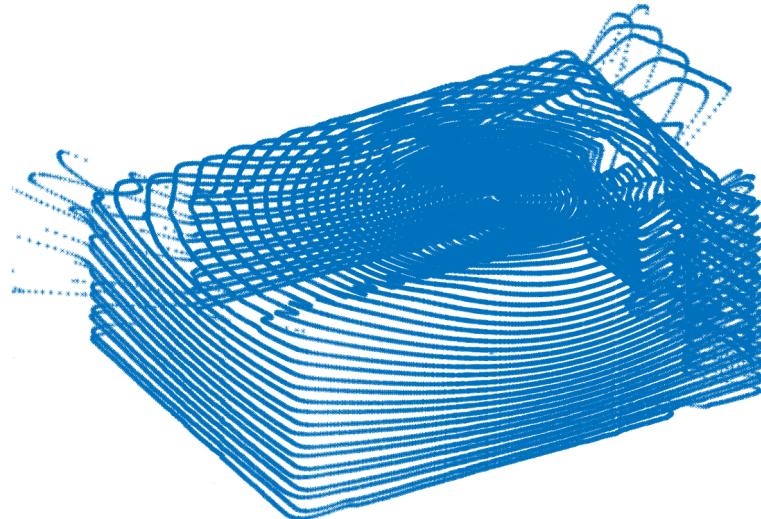


Abbildung 10.3: Darstellung mit Asterisken

Die letzte Möglichkeit ist die Darstellung mit kleinen Koordinatenpunkten. Räume können bis ins sehr kleine Detail dargestellt werden und man hat trotz vieler Messpunkte und hoher Auflösung keine Stellen mit überladener Punkteanzahl.

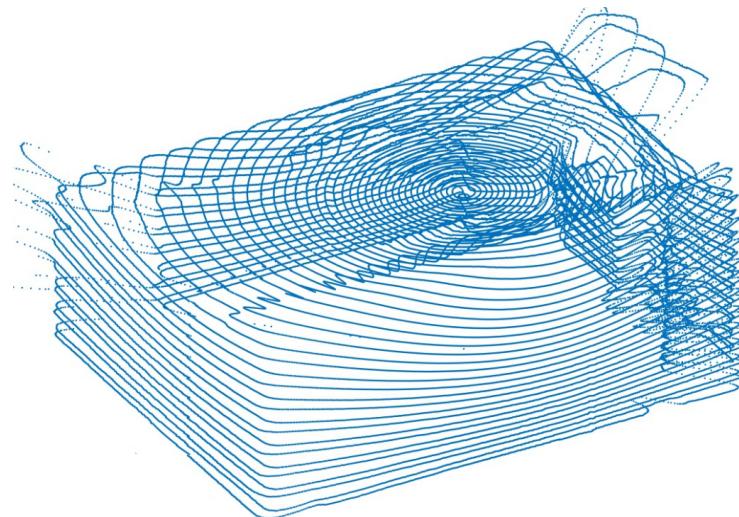


Abbildung 10.4: Darstellung mit Punkten

Aufgrund der genannten Vor- und Nachteile wird in den meisten Fällen die Darstellung mit Punkten bevorzugt.

Matlab skaliert die Achsen der Darstellung automatisch, weshalb teilweise schlecht auswertbare Bilder entstehen. Diese sind nur mit manueller Nachbearbeitung passend einstellbar. Um diese zusätzliche Arbeit zu automatisieren und die Darstellung einheitlich zu realisieren,

werden die Achsen manuell mit der Funktion "axis" skaliert. Die ersten beiden Werte geben die Skala der x-Achse in Zentimetern an. Der zweite und dritte den Wert der y-Achse. Die letzten beiden Werte den Bereich der z-Achse. Mit der Funktion "pbaspect" wird zudem die relative Größe der Achse in der späteren Darstellung festgelegt, um Verzerrungen zu vermeiden.

```
1 axis([-400 400 -400 400 0 240])
2 pbaspect([1 1 0.3])
```

Listing 10.4: Skalieren der Achsen

# 11 Validierung des Systems

Nachdem das Aufnehmen und Darstellen von Räumen funktioniert, wird die Genauigkeit des Systems untersucht. Zudem werden Versuche zu verschiedenen Auflösungen und unterschiedlicher Sensoren durchgeführt.

## 11.1 Genauigkeit des Systems

Um die Genauigkeit des Systems zu überprüfen, wird ein Raum mit dem Lidar-System vermessen. Zudem wird der Raum händisch vermessen und der Grundriss mit der Software "Sweet Home 3D" erstellt. Anschließend wird die 3D-Darstellung mit dem Grundriss des Raumes und weiterer markanter Gegenstände verglichen. Bei dem Raum handelt es sich um einen Flur mit vielen Ecken, Türen und Gegenständen. Dadurch erhält man viele verschiedene Maße, die überprüft werden können. Der Grundriss des Raumes ist in Abbildung 11.1 dargestellt. Alle weiteren Maße des Raumes können direkt in der Software abgerufen werden.

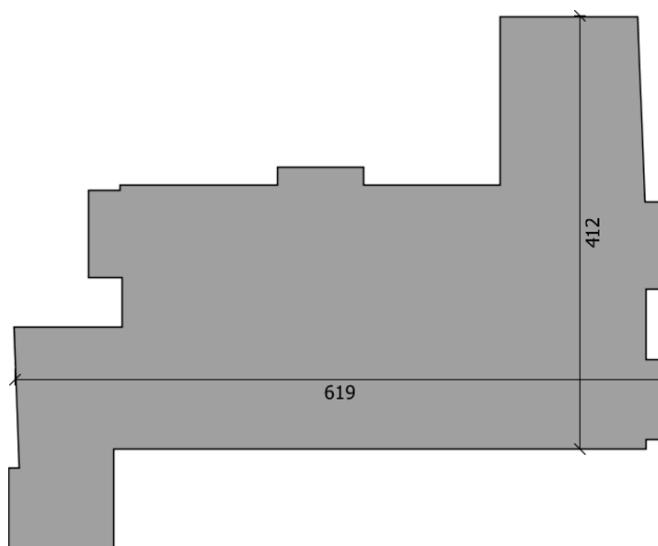


Abbildung 11.1: Grundriss des Testraums

Das Lidar-System wird im Raum aufgestellt. Die Position ist annähernd mittig und wird zudem bestimmt und in der Software eingetragen. Durch die Funktionsweise von Lidar Sensoren entstehen Schatten. So können beispielsweise Konturen hinter einer Wand, die die Lichtstrahlen reflektiert nicht detektiert werden. Diese Schatten werden ebenfalls im Grundriss eingezeichnet, um den Vergleich besser durchführen zu können. Die weißen Stellen innerhalb des Grundrisses in Abbildung 11.2 stellen diese Schatten dar.

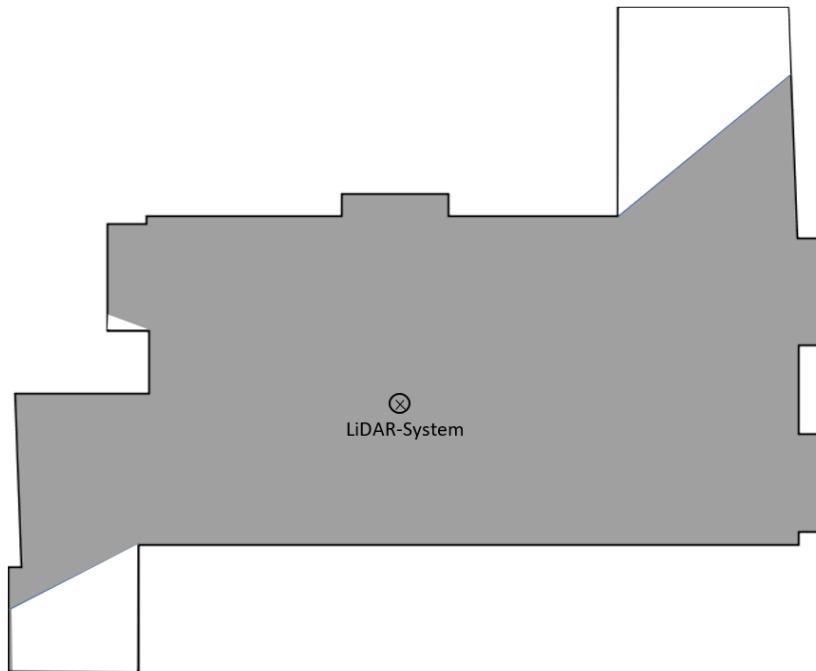


Abbildung 11.2: Grundriss des Testraums

Zum Vergleich wird nun der Grundriss benötigt, den das Lidar-System erstellt hat. Dazu wird die 3D Darstellung nur in z-Richtung betrachtet. Man erhält die Vogelperspektive des Raumes, bei dem der Grundriss auszumachen ist.

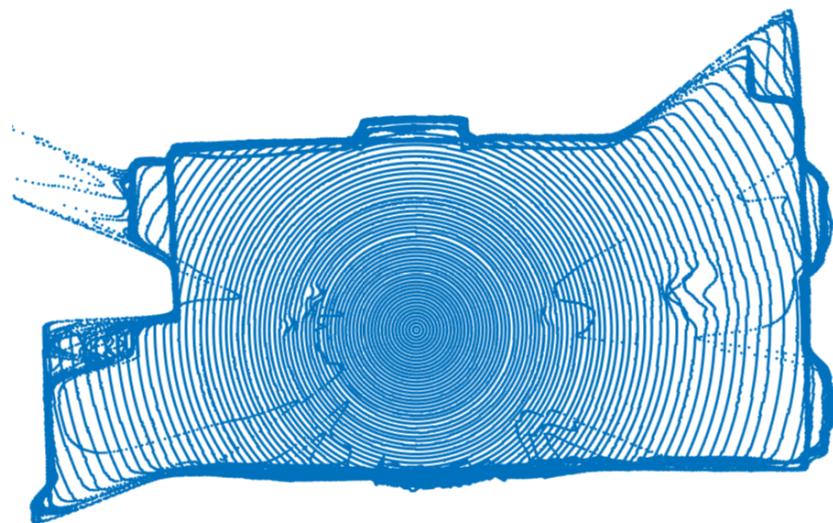


Abbildung 11.3: Vogelperspektive des Testraums

Zum grafischen Vergleich werden manuell erstellter Grundriss und die Vogelperspektive des Testraums mit einem Bildbearbeitungsprogramm im gleichen Maßstab übereinander gelegt.

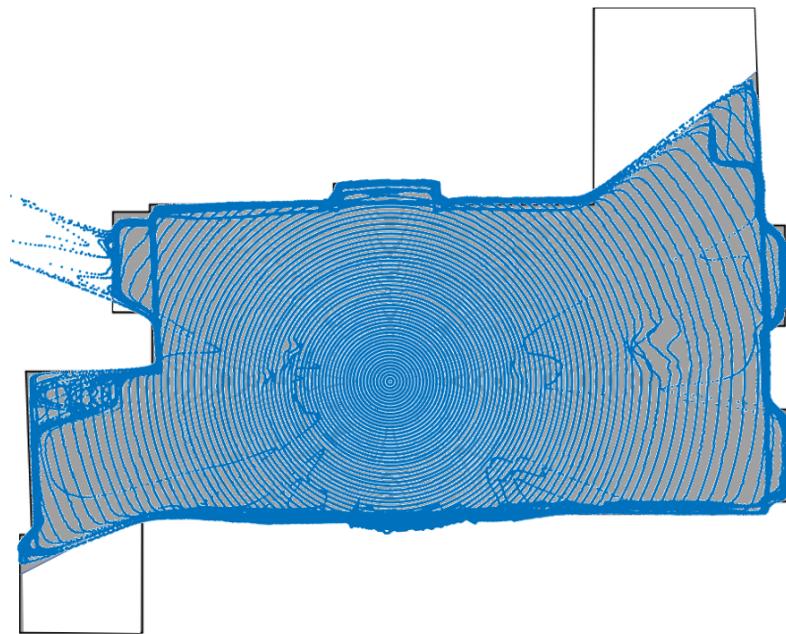


Abbildung 11.4: Grafischer Vergleich der Grundrisse

*TODO: Auswertung beschreiben, was kann man sehen? TODO: Linien an Seite raus weil Glas TODO: Höhe wird überprüft, Schrank, Bild usw, Messen mit Cursor*

## 11.2 Vergleich verschiedener Auflösungen

Durch das Einstellen verschiedener Schrittweiten der Schrittmotoren können unterschiedliche Auflösungen und Punkteverteilungen eingestellt werden. Derselbe Raum wird unter den gleichen Randbedingungen mit drei unterschiedlichen Einstellungen vermessen. Dabei bleibt sowohl die Position des Lidar-Sensors als auch der Sensor selbst gleich. Verändert wird sowohl die horizontale- als auch die vertikale Schrittweite. Dies kann im Code durch das Ändern weniger Parameter realisiert werden.

Bei den verschiedenen Auflösungen werden vor allem das Ergebnis und die benötigte Zeit zum Aufnehmen der Messdaten verglichen. Zudem soll dadurch eine Einstellung gefunden werden, die einen guten Kompromiss zwischen Auflösung und benötigter Zeit darstellt.

Als Sensor wird der "TF Mini Lidar" verwendet.

### 11.2.1 Übersicht über die Dauer, Auflösung und Anzahl an Messpunkten

Die horizontale Auflösung wird im Code in achtel Schritten des Schrittmotors angegeben. Der Motor läuft im Achtelschrittbetrieb. Für eine gesamte Umdrehung des Motors werden 200 Vollschritte benötigt. Zudem entspricht die Übersetzung von Schrittmotor zur drehbaren Basis des Lidar-Systems 1:6. Es werden also 9600 Achtelschritte benötigt, um die Basis einmal um 360 Grad zu drehen. Die Auflösung in Grad bezogen auf die Angabe im Code kann mit Formel 11.1 berechnet werden.

$$d = \frac{360}{200 \cdot 8 \cdot 6} \cdot x \quad (11.1)$$

$x$  = Anzahl Achtelschritte des Motors (in Software) [°]

$d$  = reale Drehung des Sensors in horizontaler Richtung [°]

Tabelle 11.1: Übersicht verschiedene Auflösungen

	Auflösung gering	Auflösung mittel	Auflösung hoch
<b>Auflösung horizontal [°]</b>	1,2	0,15	0,15
<b>Auflösung vertikal [°]</b>	14,4	7,2	3,6
<b>Anzahl Messpunkte</b>	7524	120049	240099
<b>Dauer [min]</b>	2	31	62

Die vertikale Grundeinheit sind Viertelschritte. Um den Sensor um die maximalen 90 Grad drehen zu können, werden 50 Vollschritte benötigt. Der Sensor ist direkt mit der Welle des Motors verbunden, wodurch keine Konstante für eine Übersetzung benötigt wird.

$$d = \frac{360}{50 \cdot 4} \cdot x \quad (11.2)$$

$x$  = Anzahl Viertelschritte des Motors (in Software) [°]

$d$  = reale Drehung des Sensors in horizontaler Richtung [°]

Die Menge der Messpunkte lässt sich über die Anzahl der horizontalen Messpunkte multipliziert mit der Anzahl der vertikalen Reihen berechnen.

Die benötigte Zeit ist linear zu der Anzahl der Messpunkte.

## 11.2.2 Vergleich der unterschiedlichen Auflösungen

Im Anschluss werden die 3D-Darstellungen grafisch verglichen und beurteilt. Vor allem die Darstellung von Details und die Richtigkeit der Maße wird überprüft.

Um den Vergleich zu erleichtern, werden Bilder aus der jeweils gleichen Perspektive erstellt und verglichen. Der Maßstab ist bei jeder Darstellung identisch.

Die Aufnahme mit niedriger Auflösung besteht aus rund 7500 Bildpunkten. Die horizontale Auflösung ist mit 1,2 Grad Abstand zwischen zwei Punkten für nicht weit entfernte Gegenstände ausreichend. Bei größerer Entfernung wie z.B. in den Ecken des Raumes wird diese Auflösung an der Wand jedoch relativ schlecht und Details werden nicht mehr erkannt. Die vertikale Auflösung von 14,4 Grad ist deutlich zu gering, um Details erkennen zu können. Türrahmen oder ähnliche größere Unebenheiten an der Wand sind nur grob auszumachen.

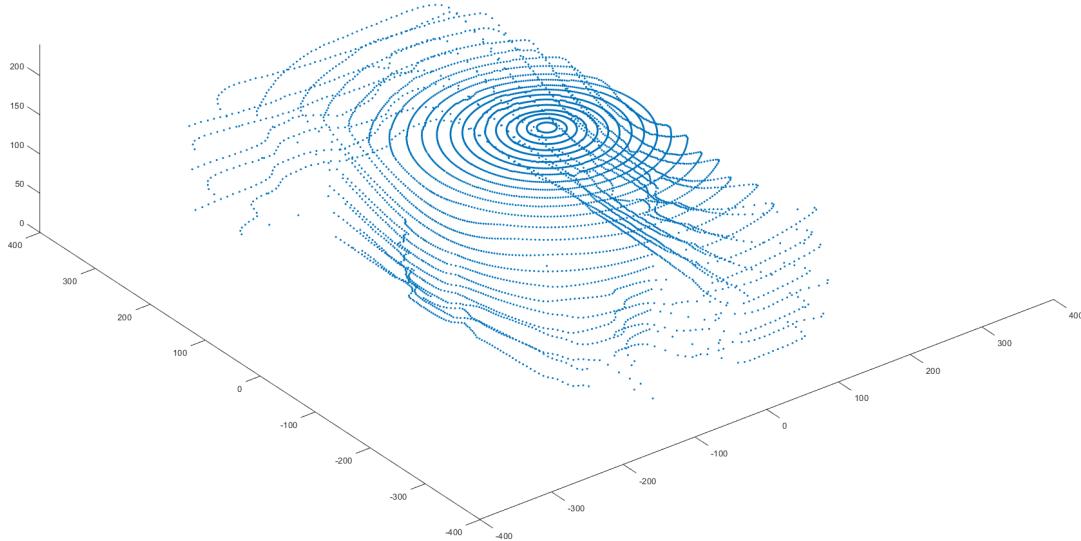


Abbildung 11.5: Niedrige Auflösung

Bei der mittleren Auflösung werden etwa 16 mal so viele Bildpunkte aufgenommen wie bei der Messung mit geringer Auflösung. Die vertikale Schrittweite wird im Vergleich zur ersten Messung halbiert, die horizontale beträgt mit 0,15 Grad ein Achtel der ursprünglichen Schrittweite.

Die horizontalen Punkte verschmelzen zu einer Linie. Dies ist ein Zeichen dafür, dass die horizontale Auflösung von 0,15 Grad für den Testraum ausreichend ist. Durch die geringere vertikale Schrittweite erhält man doppelt so viele Messebenen. Dadurch werden Details wie beispielsweise Türrahmen, Lampen und weitere Gegenstände besser erkennbar.

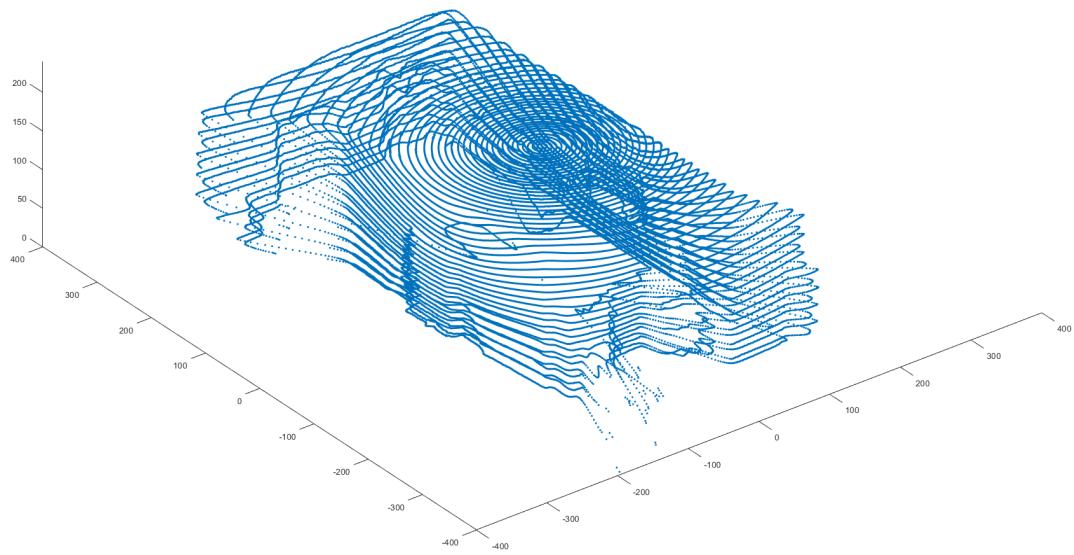


Abbildung 11.6: Mittlere Auflösung

Wie die Messung mit mittlerer Auflösung bereits gezeigt hat, reicht eine horizontale Schrittweite von 0,15 Grad bei der Größe des Testraums aus. Für den Test mit hoher Auflösung wird daher nur noch die vertikale Schrittweite halbiert. Dadurch verdoppelt sich die Anzahl der Bildpunkte. Die Messung benötigt jedoch ungefähr doppelt so lange.

Das Ergebnis ist eine 3D-Aufnahme, bei der sowohl horizontale als auch vertikale Auflösung gut ausreicht, um Details erkennen zu können.

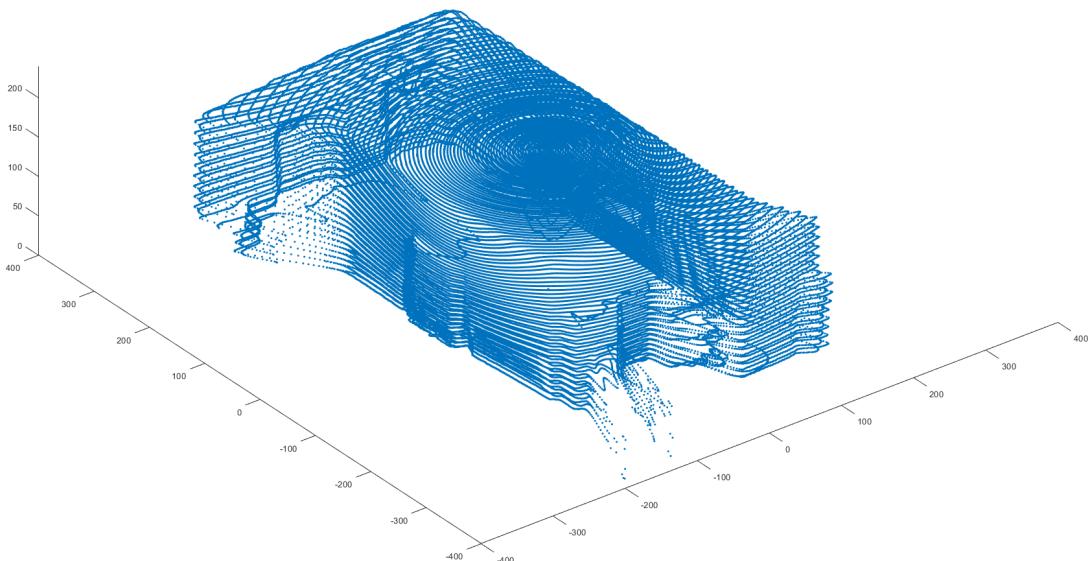


Abbildung 11.7: Hohe Auflösung

Weiter werden die erstellten Grundrisse mit niedriger, mittlere und hoher Auflösung mit dem tatsächlichen Grundriss verglichen.

Der Vergleich zeigt, dass die Maße immer bis auch geringe Abweichungen mit dem tatsächlichen Grundriss übereinstimmten. Dabei macht die Auflösung keinen Unterschied.

Deutlich erkennbar ist jedoch, wie in Abbildung 11.8 zu sehen ist, dass bei niedriger Auflösung Details wie Türrahmen verschwimmen. Zudem sind die einzelnen Linienebenen leicht zueinander verschoben und nicht eindeutig zuordenbar. Im Vergleich dazu sind bei der hohen Auflösung (Abbildung 11.9) klare Grundrisslinien erkennbar.

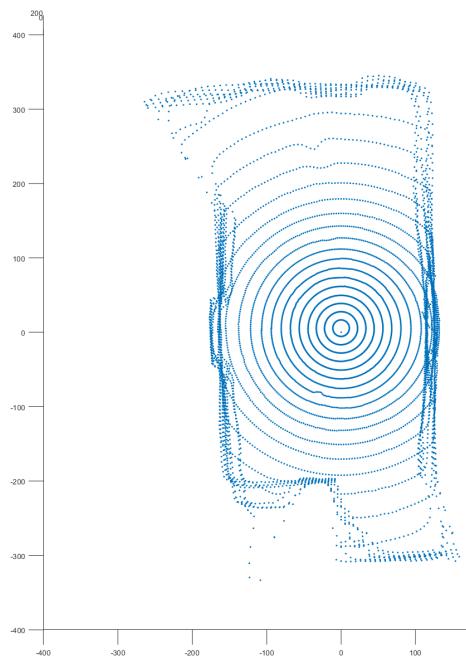


Abbildung 11.8: Vogelperspektive niedrige Auflösung

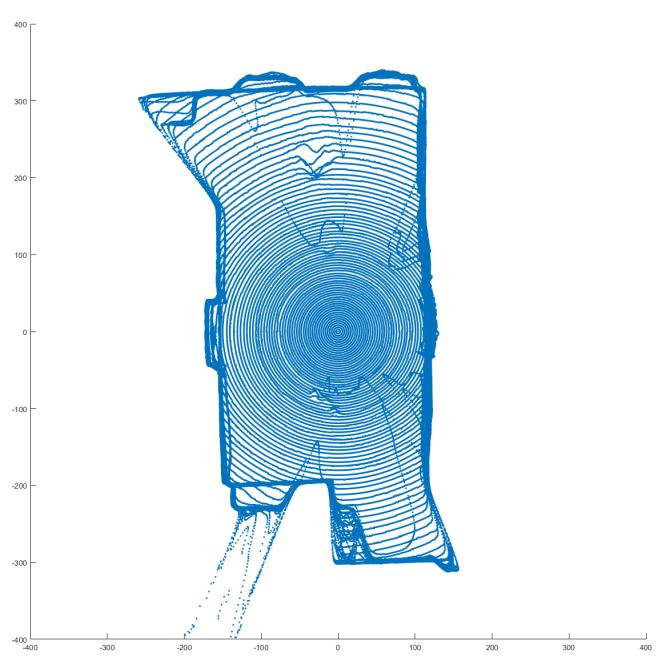


Abbildung 11.9: Vogelperspektive hohe Auflösung

Im Großen und Ganzen lässt sich sagen, dass je nach Anforderung an das System die Parameter dementsprechend eingestellt werden müssen. Möchte man nur einen groben Überblick über den Raum, reicht ein schneller Scan mit niedriger Auflösung vollkommen aus. Möchte man hingegen eine detaillierte Darstellung, bei der man auch Details virtuell vermessen kann, wird eine hohe Auflösung benötigt. Dafür dauert solch eine Messung im Vergleich zur schnellen Messung recht lange. Man sollte somit immer einen Kompromiss zwischen benötigter Auflösung und der Zeit für die Messung finden.

## 11.3 Vergleich der Sensoren

Nachdem sowohl die Mechanik als auch der erste Lidar Sensor getestet und validiert wurden, soll ein kostengünstigerer Sensor getestet werden. Dazu werden zwei Aufnahmen mit den gleichen Einstellungen aber verschiedenen Sensoren Aufgenommen. Die erste Messung wird mit dem Sensor "TF MINI" gemacht. Als zweiter Sensor wird ... genommen.

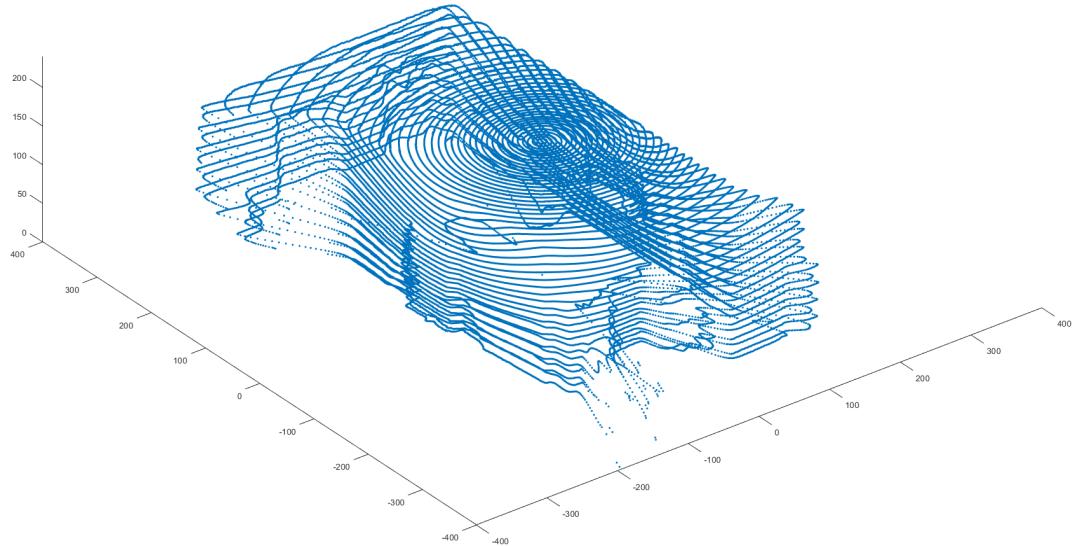


Abbildung 11.10: TF MINI

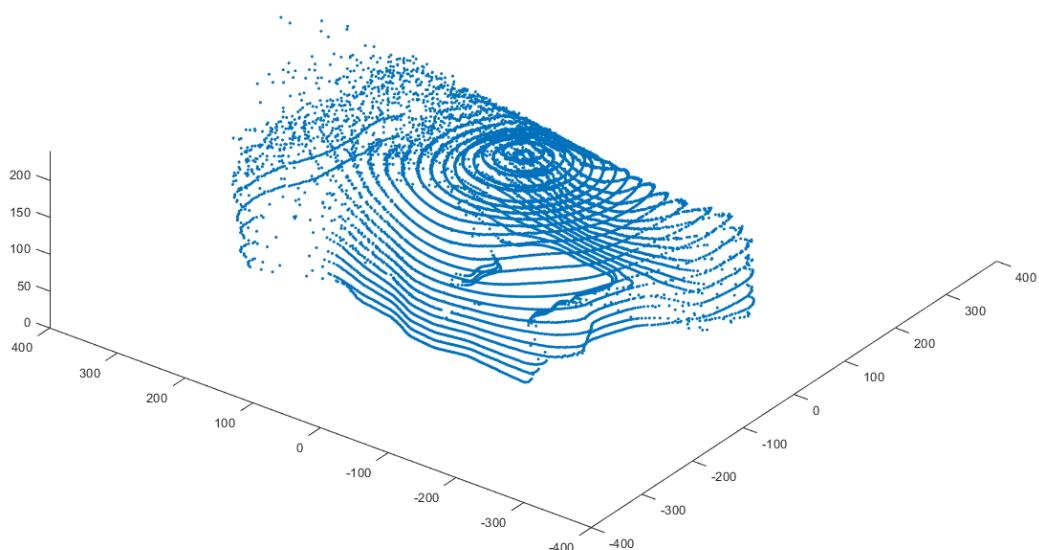


Abbildung 11.11: Sensor VLX

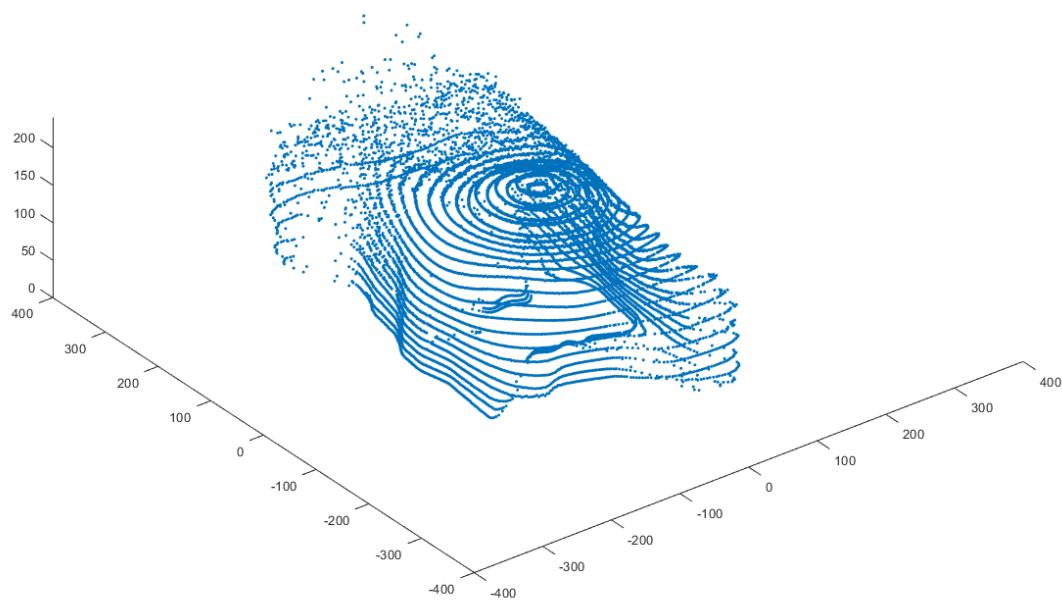


Abbildung 11.12: Sensor VLX

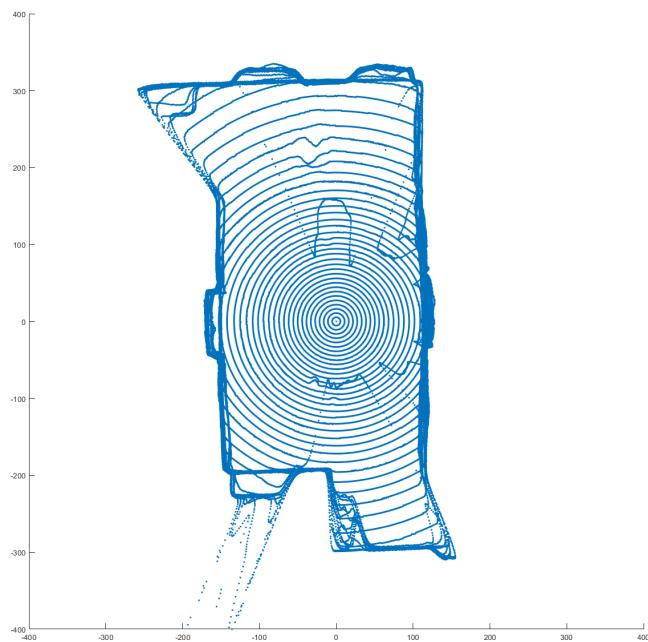


Abbildung 11.13: Beispielbild

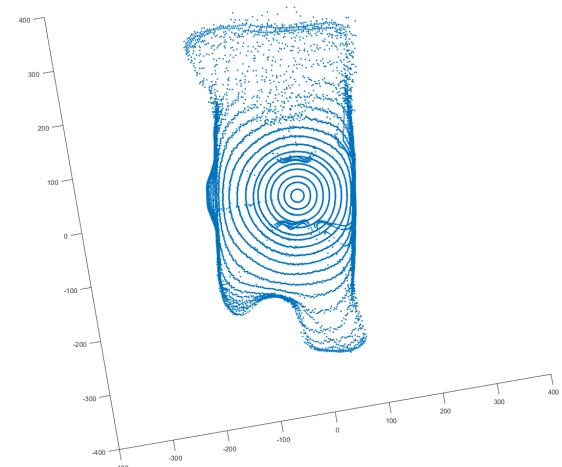


Abbildung 11.14: Beispielbild

*TODO: Aktuelle Bilder einfügen + Text*

## 12 Fazit/Zusammenfassung

Ziel dieser Arbeit war es, ein kostengünstiges System zu entwickeln, welches nach einem Scan einen Raum als 3D Punktwolke darstellen kann. Die Handhabung soll möglichst benutzerfreundlich sein. Zudem wurde eine Machbarkeitsstudie zum Entwickeln eines eigenen Sensors durchgeführt.

Das Ergebnis der Machbarkeitsstudie ist, dass der Eigenbau eines LIDAR-Sensors mittels Laserdiode und einfacher Photodiode nicht realisierbar ist. Nach wenigen Experimenten und Gesprächen mit Experten ist klar, dass ein Sensor ausgewählt und gekauft werden muss.

Im Großen und Ganzen funktioniert das im Rahmen dieser Studienarbeit entwickelte und gebaute LIDAR-System. Nach einem Scan des Raums können die Daten mithilfe eines Matlab Programms als 3D Punktwolke dargestellt werden. Das Drehen, Zoomen und Durchfliegen des Raumes funktioniert. Messungen jeglicher Distanzen direkt am Modell sind ebenfalls möglich. Auch die Auflösung entspricht und übertrifft die vorher definierten Anforderungen.

### *TODO: Werte angeben?*

Die Anforderung der Benutzerfreundlichkeit konnte aus Zeitgründen nicht optimiert werden. Das Starten des Programms muss über einen Konsolenbefehl direkt am Raspberry Pi durchgeführt werden. Änderungen der Auflösung sowie anderer Parameter müssen direkt im Code angepasst werden.

Da die Visualisierung über Matlab realisiert wird, müssen die Rohdaten per USB-Stick manuell exportiert werden. Zudem funktioniert die automatische Kalibrierung der horizontalen Achse nicht.

Ein weiterer Punkt, der gegen die Benutzerfreundlichkeit spricht, ist die benötigte Zeit für den Scan eines Raumes. Bei sehr hoher Auflösung dauert ein Scan ca. eine Stunde. Der limitierende Faktor an dieser Stelle ist die Aufnahmefrequenz des verwendeten Sensors. Sensoren mit höherer Messfrequenz sind deutlich kostenintensiver.

Im folgenden Kapitel sind zahlreiche Verbesserungsmöglichkeiten des Systems aufgeführt.  
Diese dienen zum hauptsächlich dazu, die Benutzerfreundlichkeit verbessern.

*TODO: Kapitel: Fazit überarbeiten*

# 13 Ausblick

Im folgenden sollen Aspekte aufgeführt und erläutert werden, welche am LIDAR System noch verbessert oder erweitert werden können. Teilweise sind für die Aspekte bereits Materialien vorhanden oder es wurden Verschiedene Methoden recherchiert. Alle unterlagen, welche mit dem Projekt verknüpft sind, sind im Github zu finden.

Die Verbesserungen werden in zwei Verschiedene Bereiche geteilt. Im Hardware Teil wird darauf eingegangen, welche Punkte an der Mechanik und/oder den Elektrischen Komponenten verbessert werden kann. Im zweiten Teil, der Software wird dann darauf eingegangen, wie man das System nutzerfreundlicher gestalten könnte.

## 13.1 Hardware

### 13.1.1 Platine

Da beim Entwurf der Platine einige Fehler passiert sind, diese sind bereits im Layout behoben und die Platine muss neu gefräst werden. Bevor dies allerdings geschieht sollte diese auch neu gelayoutet werden, da es verschiedene Optimierungsmöglichkeiten gibt.

Die erste Optimierungsmöglichkeit für das neue Platinenlayout ist, dass ein Zusätzlicher Pin für die Lichtschranke heraus geführt wird, damit ist die Platine wieder so flexibel einsetzbar wie gedacht. Da dann auch ein Sensor welcher mittels SPI angesteuert wird wieder einsetzbar ist.

Ein größerer Aspekt welcher im neuen Platinenlayout beachtet werden sollte ist, dass der Flachbandkabelstecker zum Raspberry Pi an den Rand zum Raspberry Pi hin umpositioniert wird. Dies erleichtert die Montage / Demontage der Platine oder des Raspberry Pi's. Zudem hat der Motortreiber des Motors 1 dann ausreichend Platz nach oben um entstehende Hitze abzuführen. Außerdem sind die gesamten Pins, welche vom Flachbandkabel verdeckt werden dann einfacher zu erreichen, dies bedeutet eine einfachere Wartung des LIDAR Systems.

### 13.1.2 Gyrosensor

Im ursprünglichen Konzept des LIDAR Systems war vorgesehen, dass sich sowohl die Horizontal- als auch die Vertikalachse selbstständig Kalibrieren können. Für die Horizontalachse hat dies durch die Verwendung einer Lichtschranke reibungslos geklappt, die Vertikalachse sollte sich mittels eines Gyrosensors in Nulllage, oder jede beliebige andere Lage, bringen können. Allerdings konnte dies im Rahmen der Studienarbeit nichtmehr implementiert werden, da der Sensor in ersten Tests zu ungenau war und weitere Tests nichtmehr möglich waren.. Der Sensor *TODO: Name Sensor* ist bereits vorhanden und muss lediglich getestet und implementiert werden.

### 13.1.3 Schleifring

Ebenfalls war im Konzept und der CAD Zeichnung des LIDAR Systems die Verwendung eines Schleifrings geplant, da bei Verwendung kein umdrehen nach  $360^\circ$  nötig ist, sondern sich das System kontinuierlich in eine Richtung drehen kann.

Da es bei dem Schleifring allerdings zu Lieferschwierigkeiten kam, konnte dieser im Rahmen der Studienarbeit nicht verbaut werden. Die Bauteile sind allerdings so konstruiert, dass der vorgesehene Schleifring lediglich eingebaut werden müsste.

### 13.1.4 Motortreiber

Die Motortreiber welche im Rahmen der Studienarbeit verwendet wurden können maximal achtel Schritte tätigen. Wenn ein weiteres erhöhen der Messauflösung des Systems gewünscht ist, könnte man bessere Motortreiber verwenden, welche in der Lage sind sechzehntel Schritte zu tätigen. Außerdem sollte der Motortreiber des Motors 1 also der Vertikalen Achse kontrolliert werden, da wie bereits erwähnt dieser bei korrekter Ansteuerung lediglich viertel Schritte tätigt.

### 13.1.5 Bedienfeld

Ein weiterer Aspekt welcher bereits teilweise vorbereitet ist, ist die anbringung eines Bedienfelds an der front des LIDAR Systems. Die bringt den Vorteil, dass die Messung nichtmehr über einen Computer gestartet werden muss, sondern das LIDAR System alleinstehend verwendbar ist.

Mögliche Elemente, welche sowohl in Hard- als auch in Software erstellt werden müssen sind:

- LCD Panel zur Ausgabe der Menüoptionen und des Fortschritts
- Drehencoder zur navigation im Menü
- Anbringen der Status LED's
- Anbringen des Ein- & Ausschalters des LIDAR Systems

Zum anbringen des Bedienfelds wurden bereits Nutenstein im vorderen Teil des Rahmens eingebbracht, so dass man das Bedienfeld einfach anbringen kann.

## 13.2 Software

Damit einige der erwähnten Hardware Implementationen möglich sind müssen auch Anpassungen an der Software vorgenommen werden.

### 13.2.1 Steuerung mit Übergabeparameter

Um eine noch einfachere Steuerung des Systems zu ermöglichen können in Zukunft die Klassen so umgeschrieben werden, dass die Übergabe von Parametern möglich ist.

Eine möglicher aufruf eines Programms könnte dann wie folgt aussehen (Listing 13.1). Diese Funktion soll den Motor, welcher durch den ersten Übergabeparameter festgelegt wird um die Anzahl Schritte welche vom zweiten Übergabeparameter festgelegt werden drehen. Die Richtung soll dabei durch das Vorzeichen des zweiten Übergabeparameters bestimmt werden. Dabei soll ein positives Vorzeichen eine Drehung mit dem Uhrzeigersinn und ein negatives Vorzeichen eine Drehung gegen den Uhrzeigersinn bewirken.

```
1 python LIDAR_Bewegen.py 1 -400
```

Listing 13.1: Beispiel Aufruf einer Python Funktion mit Übergabeparametern

Um diese Funktion zu implementieren müssen allerdings die Bestehenden Funktionen überarbeitet werden. In Listing 13.2 ist ein Code Beispiel für ein Programm welches über Übergabeparameter die Motoren steuern kann. Dieses Beispiel ist allerdings noch nicht am System selbst getestet worden.

```
1 ## Programm zum Bewegen eines Motors
2
3 #Bibliotheken
4 import sys
5
6 #Eigene Dateien
7 import Motor
8
9
10 # Motor 1, Nema 11
11 M1 = Motor.MOTOR(31,29,37,35,33)
12
13 # Motor 2, Nema 17
14 M2 = Motor.MOTOR(18,16,36,38,40)
15
16 if(len(sys.argv) < 3):
17     print("""Aufruf wie folgt:
18         python LIDAR_Bewegen.py <nummerMotor> <Schritte>
19         <nummerMotor> = 1 oder 2
20         <Schritte> = positiv fuer Uhrzeigersinn, negativ fuer gegen den
21             Uhrzeigersinn
22     """)
23 else:
24     m = sys.argv[1]
25     s = sys.argv[2]
26     dir = True
27     if(s > 0):
28         dir = True
29     else if(s < 0):
30         dir = False
31     s = s * -1
32     else:
33         print("Bitte Schritte angeben")
34
35     if(m == 1):
36         M1.moveMotor(dir, s, 0.001)
37     else if(m == 2):
38         M2.moveMotor(dir, s, 0.001)
39     else:
39         print("Bitte Motor angeben")
```

Listing 13.2: Python Beispiel Funktion welche Übergabeparamenter akzeptiert und ausführt

Um die gewünschte Funktion zu implementieren ist die Bibliothek 'sys' sehr wichtig, denn diese Stellt die übergebenen Werte in einem Array zur Verfügung. Danach ist das Programm recht einfach aufgebaut, da in Zeile 7 - 14 die Motorklasse importiert und die beiden Motoren Initialisiert werden. Bevor mit dem eigentlichen ausführen des Programms, bzw. der Bewegung des Motors begonnen wird, wird überprüft ob mindestens 2 Werte übergeben wurden. Falls nicht wird eine Ausgabe darauf hinweisen und die Verwendung erläutern.

Anschließend wird ab Zeile 23 damit begonnen die übergebenen Werte in lokale Variablen zu übernehmen und die angegebene Richtung, welche beim Aufrufen durch das Vorzeichen des zweiten Parameters festgelegt wird, zu prüfen. Danach wird nur noch die Bekannte Funktion der Motor Klasse zum bewegen des Motors aufgerufen.

### 13.2.2 Bedienfeld Statusausgabe und Steuerung

Um ein Bedienfeld zu realisieren muss gegebenenfalls ein Komplettes Menü erstellt werden, durch welches die verschiedenen Funktionen aufrufbar sind. Bei der Implementierung gilt es darauf zu achten, dass die Objektorientierte Programmierung beibehalten wird und die Möglichkeit weitere Funktionen zu implementieren erhalten bleibt.

Um heraus zu finden wie weit eine Messung fortgeschritten ist, muss lediglich im Programmablauf beobachtet werden, wie viele der Festgelegten Messpunkte bereits aufgenommen wurden. Dies kann relativ einfach über die beiden Zähler Variablen der while Schleifen realisiert werden. *TODO: Recherche Python console Ladebalken und Pseudocode erstellen*

### 13.2.3 Webinterface

Die selbe Steuerung welche über das Bedienfeld direkt am System möglich ist, kann auch am PC in einem ansprechenden Graphical User Interface (GUI) möglich sein. Dazu ist die Idee, dies mittels einem Webinterface zu realisieren. Der Raspberry Pi könnte dazu ein eigenständiges Wireless Local Area Network (WLAN) verwalten. Für die Darstellung der Website kann beispielsweise ein NodeJS Server auf dem Raspberry Pi aufgesetzt werden. Um von javascript anfragen an den Server, bzw. Python zu senden kann die Bibliothek Flask verwendet werden. Im folgenden wird auf die Ideen zu den einzelnen Komponenten näher eingegangen.

## WLAN auf dem Raspberry Pi einrichten

Da im Projekt ein Raspberry Pi der dritten Generation verwendet wurde, besitzt dieser durch den verbauten WLAN-Chip die Möglichkeit ein eigenes WLAN-Netzwerk zu erstellen und verwalten. <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/> Durch einige Änderungen in den Netzwerkeinstellungen lässt sich diese Funktion nutzen. Zunächst müssen dafür die Packages zur Verwaltung der Zugriffe auf das Netzwerk installiert werden.

```
1 sudo apt-get install dnsmasq hostapd
```

Listing 13.3: Installation dnsmasq hostapd

Diese neu installierten Packages müssen anschließend auch konfiguriert werden. Dazu wird folgende Datei aufgerufen und durch eine Zeile ergänzt.

```
1 Sudo nano /etc/dhcpcd.conf
```

Listing 13.4: Konfiguration DHCP Server

Wird ergänzt durch

```
1 Denyinterfaces wlan0
```

Listing 13.5: Konfiguration DHCP Server

Damit der Raspberry Pi dann auch als Bereitsteller eines Netzwerks erkannt werden kann und immer wieder wird bekommt der Raspberry Pi eine Statische IP! (IP!)-Adresse.

```
1 Sudo nano /etc/network/interfaces
```

Listing 13.6: Konfiguration Interfaces

Wird dabei um mehrere Zeilen, welche zum Einstellen der Statischen IP!-Adresse dienen ergänzt.

```
1 allow-hotplug wlan0
2 iface wlan0 inet static
3 address 192.168.0.1
4 netmask 255.255.255.0
5 network 192.168.0.0
6 broadcast 192.168.0.255
```

---

Listing 13.7: Konfiguration Interfaces

Anschließend muss der dhcpcd client und der WLAN-Chip neugestartet werden.

```
1 Sudo service dhcpcd restart
2 sudo ifdown wlan0; sudo ifup wlan0
```

Listing 13.8: Konfiguration Interfaces

Danach kann das Package, hostapt ebenfalls konfiguriert werden.

```
1 Sudo nano /etc/hostapd/hostapd.conf
```

Listing 13.9: Konfiguration Hostapd

Dazu müssen folgende Zeilen in diese Datei geschrieben werden.

```
1 # Schnittstelle und Treiber
2 interface=wlan0
3 driver=nl80211
4 # WLAN-Konfiguration
5 ssid=LIDAR_WLAN
6 channel=1
7 hw_mode=g
8 ieee80211n=1
9 ieee80211d=1
10 country_code=DE
11 wmm_enabled=1
12 # WLAN-Verschlüsselung
13 auth_algs=1
14 wpa=2
15 wpa_key_mgmt=WPA-PSK
16 rsn_pairwise=CCMP
17 wpa_passphrase=
```

Listing 13.10: Konfiguration Hostapd

Anschließend muss lediglich eine weitere Zeile ergänzt werden, damit die Konfiguration vollständig ist.

```
1 Sudo nano /etc/default/hostapd
```

Listing 13.11: Konfiguration Hostapd

Dort muss folgende Zeile ergänzt werden

```
1 DAEMON_CONF="/etc/hostapd/hostapd.conf"
```

Listing 13.12: Konfiguration Hostapd

Dann kann das letzte Package dnsmasq eingerichtet werden. Dieses Package ist dafür zuständig, die **IP!**-Adressen an die Nutzer des WLAN zu verteilen. Da die Grundkonfiguration von dnsmasq sehr viele Einstellungen beinhaltet sollte diese abgespeichert werden bevor eine neue eigene Konfigurationsdatei erstellt wird.

```
1 sudo mv /etc/dnsmasq.conf /etc/dnsmasq.conf_alt
2 sudo nano /etc/dnsmasq.conf
```

Listing 13.13: Konfiguration dnsmasq

Anschließend können folgende Zeilen in die neue Konfigurationsdatei eingetragen werden.

```
1 Interface=wlan0
2 no-dhcp-interface=eth0
3 listen-address=192.168.0.1
4 bind-interfaces
5 server=8.8.8.8
6 dhcp-range=192.168.0.50,192.168.0.150,240h
```

Listing 13.14: Konfiguration dnsmasq

Damit wenn der Raspberry Pi über **LAN!** (**LAN!**) an ein Netzwerk angeschlossen ist auch ein Internetzugriff stattfinden kann müssen die Pakete auch weitergeleitet werden, dazu sind weitere Einstellungen notwendig.

```
1 Sudo nano /etc/sysctl.conf
```

Listing 13.15: Konfiguration IPV4

Muss dazu mit der zeile

```
1 Net.ipv4.ip_forward=1
```

Listing 13.16: Konfiguration IPV4

Danach muss der Raspberry Pi Neu gestartet werden. Anschließend müssen folgende Zeilen für die Weiterleitung Ausgeführt werden.

```
1 sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
2 sudo iptables -A FORWARD -i eth0 -o wlan0 -m state --state RELATED,
   ESTABLISHED -j ACCEPT
3 sudo iptables -A FORWARD -i wlan0 -o eth0 -j ACCEPT
4 sudo sh -c "iptables-save > /etc/iptables.ipv4.nat"
```

Listing 13.17: Konfiguration IPV4

Zuletzt muss noch eine Datei geändert werden.

```
1 Sudo nano /etc/rc.local
```

Listing 13.18: Konfiguration IPV4

Und folgende Zeile eingefügt werden

```
1 Iptables-restore < /etc/iptables.ipv4.nat
```

Listing 13.19: Konfiguration IPV4

Dann kann final der Hostapd und die dnsmasq gestartet werden.

```
1 Sudo service hostapd start
2 sudo service dnsmasq start
3 Suod reboot
```

Listing 13.20: Starten der neu installierten Packages

Nach dem Neustart des Raspberry Pi sollte ein WLAN sichtbar sein um man sollte sich mit diesem Verbinden können. <https://www.elektronik-kompendium.de/sites/raspberry-pi/2002171.htm> <https://www.randombrick.de/raspberry-pi-als-wlan-access-point-nutzen/>

## NodeJS Server auf dem Raspberry Pi einrichten

Um einen Webserver auf dem Raspberry Pi einrichten zu können wird NodeJS verwendet. Auch dies muss erst installiert und eingerichtet werden.

```
1 curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -
2 sudo apt-get install -y nodejs
```

Listing 13.21: Installation NodeJS

Ob die Installation erfolgreich war lässt sich mit folgendem Befehl überprüfen.

```
1 node -v
```

Listing 13.22: Installation NodeJS

## Flask in Javascript und Python verwenden

Flask ist ein Python Microframework, welches es ermöglicht verschiedene Routen direkt in Python über einen Webserver anzusprechen. Dies gibt in diesem Fall die Möglichkeit aus dem NodeJS Server heraus Python Programme zu starten. Um Flask zu verwenden muss dieses natürlich zunächst installiert werden. Dazu muss ein Ordner angelegt sein und in diesem Ordner eine Virtuelle Umgebung erstellt werden.

```
1 mkdir LIDAR
2 cd LIDAR
3 python3 -m venv venv
4 . venv/bin/activate
```

Listing 13.23: Installation Flask

Anschließend kann Flask installiert werden.

```
1 Sudo pip install Flask
```

Listing 13.24: Installation Flask

Nun kann eine Rote hinzugefügt werden, welche man dann, wenn der Flask Server gestartet ist über eine Route aufrufen kann. Im Fall dieses Projektes eignet sich Flask dazu die bereits vorhandenen Dateien aufrufen zu lassen. Die zugehörige Datei dazu könnte wie Folgt aussehen.

```
1 From flask import Flask
2 import subprocess
3 app=Flask(__name__)
4
5 @app.route('/movemotor1/<int: steps>')
6 def moveMotor1(steps):
7     subprocess.run(["python", "LIDAR_Bewegen.py", "1", str(steps)])
8
9 @app.route('/movemotor2/<int: steps>')
```

```
10 def moveMotor2(steps):  
11     subprocess.run(["python", "LIDAR_Bewegen.py", "2", str(steps)])  
12     return;
```

Listing 13.25: Flask Beispielprogramm

Dies ist ein ungetesteter Beispielcode um die neu entworfene Funktion zum Bewegen der Motoren aufzurufen. Dabei ist es möglich über zwei verschiedene Routen die beiden Motoren anzusprechen und über einen Zusatz in der Route kann die Anzahl der Schritte angegeben werden. <http://flask.pocoo.org>

# Anhang

## Literatur

- [1] *10mm Lichtschranke Photoelectric Infrarot Zähler Counter Sensor Modul, lalaal.*  
URL: <https://eckstein-shop.de/10mm-Lichtschranke-Photoelectric-Infrarot-Zaehler-Counter-Sensor-Modul>.
- [2] *DMOS Microstepping Driver with Translator And Overcurrent Protection.* URL: <https://www.pololu.com/file/0J450/A4988.pdf>.
- [3] TU Dresden. *Optische Triangulation zur Distanzmessung.* Okt. 2014. URL: [https://tu-dresden.de/ing/elektrotechnik/ressourcen/dateien/iee/old2\\_pmp/studium/lehre/MST/d1/Pr/triangulation?lang=en](https://tu-dresden.de/ing/elektrotechnik/ressourcen/dateien/iee/old2_pmp/studium/lehre/MST/d1/Pr/triangulation?lang=en).
- [4] Mikhantiev E. *The Structure of the silicon APD.* 2009. URL: <https://de.wikipedia.org/wiki/Datei:APD.png>.
- [5] Ulrich Fischer u. a. *Tabellenbuch Metall.* Europa Lehrmittel, 2011.
- [6] TU Ilmenau. *Lichtgeschwindigkeit.* Sep. 2018. URL: <https://www.tu-ilmenau.de/fileadmin/media/exphys1/Physikpraktikum/Anleitungen/07.pdf>.
- [7] Fraunhofer IPM. *Optische Entfernungsmeßung.* Feb. 2918. URL: <https://www.ipm.fraunhofer.de/de/gf/objekterfassung-laserscanning/komp/abstandsmessung.html>.
- [8] Nikolai Kutscher und Beate Mielke. *3D Kameras – basierend auf Lichtlaufzeitmessung.* 2005. URL: [http://www.inf.fu-berlin.de/lehre/SS05/Autonome\\_Fahrzeuge/3dKameras.pdf](http://www.inf.fu-berlin.de/lehre/SS05/Autonome_Fahrzeuge/3dKameras.pdf).
- [9] John Kvam. *Time of Flight: Principles, Challenges and Performance.* 2017. URL: [https://www.st.com/content/dam/technology-tour-2017/session-1\\_track-4\\_time-of-flight-technology.pdf](https://www.st.com/content/dam/technology-tour-2017/session-1_track-4_time-of-flight-technology.pdf).
- [10] Lasercomponents. „Positionsmessung PSD oder CCD“. In: *Applicationsreport* (Jan. 2012).

- [11] *Long-distance ranging sensor expansion board based on VL53L1X for STM32 Nucleo.*  
URL: <https://www.st.com/en/ecosystems/x-nucleo-53l1a1.html>.
- [12] Mamamatsu. *Characteristics and use of Si APD*. Mai 2004. URL: [http://neutron.physics.ucsb.edu/docs/Avalanche\\_photodiodes\\_info.pdf](http://neutron.physics.ucsb.edu/docs/Avalanche_photodiodes_info.pdf).
- [13] *MPU-6000and MPU-6050Product Specification Revision 3.4*. URL: [https://store.invensense.com/datasheets/invensense/MPU-6050\\_DataSheet\\_V3%204.pdf](https://store.invensense.com/datasheets/invensense/MPU-6050_DataSheet_V3%204.pdf).
- [14] TU München. *Inferometrie*. Feb. 2019. URL: <https://www.ph.tum.de/academics/org/labs/mw/INT.pdf>.
- [15] *NEMA ICS 16*. National Electrical Manufacturers Association, 2001. URL: <https://www.nema.org/Standards/Pages/Motion-Position-Control-Motors-Controls-and-Feedback-Devices.aspx>.
- [16] PerkinElmer. *Avalanche photodiode - a user guide*. 2010. URL: [http://www.perkinelmer.com/CMSResources/Images/44-6538APP\\_AvalanchePhotodiodesUsersGuide.pdf](http://www.perkinelmer.com/CMSResources/Images/44-6538APP_AvalanchePhotodiodesUsersGuide.pdf).
- [17] *Pololu Stepper Motor NEMA 17*. URL: <https://eckstein-shop.de/Pololu-Stepper-Motor-NEMA-17-Unipolar-Bipolar-200-Steps-Rev-4248mm-4V-12-A-Phase>.
- [18] Michael Sackewitz. „Leitfaden zur optischen 3D-Messtechnik“. In: *Frauenhofer Leitfaden 14* (2014), S. 33–34.
- [19] *Seeedstudio Grove - TF Mini LiDAR Datasheet*. URL: [https://www.google.com/urllib?sa=t&rct=j&q=&esrc=s&source=web&cd=2&ved=2ahUKEwiu1ef2\\_MLgAhUCbFAKHRp7DaQQFjABegQIAxAC&url=http%3A%2F%2Fstatics3.seeedstudio.com%2Fassets%2Ffile%2Fbazaar%2Fproduct%2FLiDAR\\_TF\\_mini\\_Datasheet.docx&usg=AOfVaw2Td3zuc0Alic0Q23PnxB2-](https://www.google.com/urllib?sa=t&rct=j&q=&esrc=s&source=web&cd=2&ved=2ahUKEwiu1ef2_MLgAhUCbFAKHRp7DaQQFjABegQIAxAC&url=http%3A%2F%2Fstatics3.seeedstudio.com%2Fassets%2Ffile%2Fbazaar%2Fproduct%2FLiDAR_TF_mini_Datasheet.docx&usg=AOfVaw2Td3zuc0Alic0Q23PnxB2-).
- [20] Spektrum. *Photodiode*. 1998. URL: <https://www.spektrum.de/lexikon/physik/photodiode/11179>.
- [21] *VL53L1X*. URL: <https://www.st.com/resource/en/datasheet/vl53l1x.pdf>.
- [22] Prof. J. Zhang. *Angewandte Sensorik*. Dez. 2003. URL: [https://tams.informatik.uni-hamburg.de/lehre/2003ws/vorlesung/angewandte\\_sensorik/vorlesung\\_06.pdf](https://tams.informatik.uni-hamburg.de/lehre/2003ws/vorlesung/angewandte_sensorik/vorlesung_06.pdf).
- [23] ETH Zürich. *Elektronische Distanzmessung*. URL: [http://webarchiv.ethz.ch/geometh-data/student/sensorik/200506\\_Vorlesungen/04\\_EDM.pdf](http://webarchiv.ethz.ch/geometh-data/student/sensorik/200506_Vorlesungen/04_EDM.pdf).

## Aufteilung der Kapitel

Tabelle A1: Aufteilung der Kapitel

Kapitel	Author
1. Einleitung	Alexander Kehrer
2.1. Photodioden	Marcel Wagner
2.2. Schrittmotoren	Alexander Kehrer
3. Grundlagen Laserentfernungsmessung	Marcel Wagner
4. Stand der Technik	Marcel Wagner
5. Machbarkeitsstudie	Marcel Wagner
6. Matlab Modell	Alexander Kehrer
7. Mechanik	Marcel Wagner
8. Hardware	Alexander Kehrer
9. Code	Marcel Wagner
10. Auswertung und Darstellung mit Matlab	Alexander Kehrer
11. Validierung des Systems	Alexander Kehrer
12. Fazit/Zusammenfassung	Alexander Kehrer
13. Ausblick	Marcel Wagner

## Github Repository

Im Rahmen der Studienarbeit wurde ein Github Repository zur Verwaltung der Daten angelegt. Da die Dateien sehr umfangreich sind wird auf ein einzelnes einfügen verzichtet und stattdessen ein Verweis auf das Repository eingebracht.

Github Repository: <https://github.com/WagnerMarcel/LIDAR>

Tabelle A2: Verweise auf Github Repository

Kapitel	Anhang
1. Einleitung	-
2. Grundlagen Elektronik	-
3. Grundlagen Laserentfernungsmessung	-
4. Stand der Technik	-
5. Machbarkeitsstudie	Protokolle zu den Versuchen
6. Matlab Modell	Matalb code zum Modell
7. Mechanik	Technische Zeichnungen
8. Hardware	Datenblätter & Platinen
9. Code	Code
10. Auswertung und Darstellung mit Matlab	Matlab Code & Bilder
11. Validierung des Systems	-
12. Fazit/Zusammenfassung	-
13. Ausblick	-