



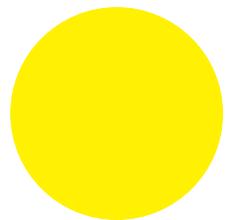
BOSCH

Technik fürs Leben



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

Seminararbeit



des Studiengangs -todo-

an der Dualen Hochschule Baden-Württemberg Stuttgart

von

-todo-

-todo-

Bearbeitungszeitraum

-todo-

Matrikelnummer, Kurs

-todo-, TEL16GR2

Ausbildungsfirma

Robert Bosch GmbH, -todo-

Betreuer

-todo-

Gutachter

Duale Hochschule Baden Würtemberg, STUTTGART

Ausbildungsbereich Technik

Fachrichtung Elektrotechnik / Informatik / Maschinenbau / Mechatronik

Bericht über die Ausbildung in der betrieblichen Ausbildungsstätte im _____ . Studienhalbjahr.

Name des Studierenden: _____

Studienjahrgang: _____

Einsatz in Abteilung: (sowohl Geschäftsbereich/Business-Unit/Abteilungsname ausgeschrieben als auch Abteilungs-Abk. entsprechend Outlook-Eintrag Betreuer)

Standort: _____

vom: _____ bis: _____

Thema: (Inhalt des Praktikums allgemeinverständlich abstrahiert, aussagefähig, prägnant, ohne Abkürzungen, wird als Tätigkeitsbeschreibung ins betriebliche Zeugnis übernommen, identisch zu Studentenportal)

Betreuer: _____

Stellungnahme des Betreuers:

Dieser Bericht wurde geprüft und ist sachlich und fachlich richtig.

Ort

Datum

Abteilung, Unterschrift

Selbstständigkeitserklärung des Studenten

gemäß §5(3) der „Studien- und Prüfungsordnung DHBW Technik“ vom 29.September 2015:

Ich habe die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Ort

Datum

Unterschrift

Selbstständigkeitserklärung

Ich versichere hiermit, dass ich meine Seminararbeit mit dem Thema: *Entwurf, Entwicklung und Validierung eines LIDAR Systems* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Stuttgart, -todo-

-todo-

Sperrvermerk

Die vorliegende Seminararbeit mit dem Titel

Entwurf, Entwicklung und Validierung eines LIDAR Systems

enthält unternehmensinterne bzw. vertrauliche Informationen der Robert Bosch GmbH, ist deshalb mit einem Sperrvermerk versehen und wird ausschließlich zu Prüfungszwecken am Studiengang -todo- der Dualen Hochschule Baden-Württemberg Stuttgart vorgelegt.

Der Inhalt dieser Arbeit darf weder als Ganzes noch in Auszügen Personen außerhalb des Prüfungsprozesses und des Evaluationsverfahrens zugänglich gemacht werden, sofern keine anders lautende Genehmigung der Ausbildungsstätte (Robert Bosch GmbH) vorliegt.

Stuttgart, -todo-

-todo-

Abstract

TODO: deutscher Abstract....

Abstract

TODO: english abstract....

Inhaltsverzeichnis

Abkürzungsverzeichnis	VIII
Abbildungsverzeichnis	IX
Tabellenverzeichnis	X
Formelverzeichnis	XI
Listings	XII
1 Grundlagen Elektronik	1
1.1 Photodioden	1
1.1.1 Avalanche Photo Diode (APD)	2
2 Grundlagen Laserentfernungsmessung	4
2.1 Lichtlaufzeitmessung	4
2.1.1 Grundprinzip	4
2.1.2 Herausforderungen	5
2.2 Phasenverschiebung	6
2.3 Triangulation	7
3 Grundlagen	8
3.1 Schrittmotoren	8
4 Matlab Modell	9
5 Mechanik	10
5.1 Anforderungen	10
5.2 Entwurf	10
5.2.1 Oberer Aufbau	10
5.2.2 Basis	12
5.2.3 Rahmen	13
5.3 Umsetzung	14
6 Hardware	16
6.1 Funktionseinheit Distanzbestimmung	16
6.1.1 TF MINi	16

6.1.2	VLX...	17
6.2	Funktionseinheit Ausrichtung des Sensors	17
6.2.1	Schrittmotoren	17
6.2.2	Schrittmotortreiber	18
6.3	Funktionseinheit Kalibrierung	19
6.3.1	Lichtschranke	19
6.3.2	Gyroensor	20
6.4	Platinen	20
6.5	Abwärtswandler	20
7	Code	21
7.1	Motor	21
7.1.1	Konstruktor	21
7.1.2	Bewegen des Motors	22
7.2	Lidar	23
7.2.1	Konstruktor und Variablen	23
7.2.2	Aufnehmen von Messdaten	24
7.3	Steuerung	25
7.3.1	Pin Definitionen und Initialisieren der Klassen	26
7.3.2	Zusätzliche Funktionen	27
7.3.3	Variablen deklaration und aufrufen von Funktionen	28
7.3.4	Erstellen der Datei zum Speichern der Daten	29
7.3.5	Schleifen der Steuerung	30
8	Auswertung und Darstellung mit Matlab	32
8.1	Importieren und Zuordnen der Messwerte	32
8.2	Umwandlung von Kugelkoordinaten zu kartesischen Koordinaten	33
8.3	Darstellung der Messwerte	35
9	Validierung des Systems	39
9.1	Genauigkeit des Systems	39
9.2	Vergleich verschiedener Auflösungen	42
9.2.1	Übersicht über die Dauer, Auflösung und Anzahl an Messpunkten	43
9.2.2	Vergleich der Ergebnisse	44
9.3	Vergleich der Sensoren	44
Anhang		A
Literatur		A

Abkürzungsverzeichnis

BSP	Board Support Package
LIDAR	Light Distance and Ranging
ToF	Time of Flight
3D	Dreidimensional
CAD	Computer Aided Design
NEMA	National Electrical Manufacturers Association
GPIO	General Purpose Input Output
APD	Avalanche Photo Diode
SPAD	Single Photon Avalanche Diode
UART	Universal Asynchronous Reciever Transmitter
I2C	Inter-Integrated Circuit
SPI	Serial Peripheral Interface
csv	comma separated values
CNC	Computerized Numerical Control

Abbildungsverzeichnis

1.1	Schematischer Aufbau einer Photodiode [8] (p^+ starke p-Dotierung)	1
1.2	Schematischer Aufbau einer APD [1] (p^+ starke p-Dotierung, $p^- (\pi)$ schwache (intrinsische) p-Dotierung, n^+ starke n-Dotierung) (1 - Metallkontakte, 2 - Entspiegelung)	2
2.1	Time of Flight (ToF) Prinzip [3]	4
5.1	Oberer Aufbau der Mechanik	11
5.2	Basis der Mechanik	12
5.3	Rahmen	13
5.4	Mechanik mit montierter Elektronik	15
6.1	TFmini	17
8.1	Kugelkoordinaten	34
8.2	Darstellung mit Linien	36
8.3	Darstellung mit Asterisken	37
8.4	Darstellung mit Punkten	37
9.1	Grundriss des Testraums	40
9.2	Grundriss des Testraums	41
9.3	Vogelperspektive des Testraums	41
9.4	Grafischer Vergleich der Grundrisse	42
9.5	Beispielbild	44
9.6	Beispielbild	44

Tabellenverzeichnis

6.1	Schrittweite	19
7.1	Beispieldatei	29
9.1	Übersicht verschiedene Auflösungen	44

Formelverzeichnis

2.1 Berechnung der Entfernung mittels Lichtlaufzeit	5
2.2 Berechnung der minimalen und maximalen Zeit	6
5.1 Berechnung Übersetzungsverhältnis [2]	14
8.1 Umrechnung Kugelkoordinaten in kartesische Koordinaten	35
9.1 Berechnung horizontiale Auflösung	43
9.2 Berechnung vertikale Auflösung	43

Listings

7.1	Bibliotheken der Motor Klasse	21
7.2	Konstruktor der Motor Klasse	22
7.3	Funktion zum Bewegen des Motors	23
7.4	Bibliotheken der Lidar Klasse	23
7.5	Kostruktor der Lidar Klasse	24
7.6	Funktion um Distanz vom LIDAR Sensor zu erhalten	24
7.7	Bibliotheken zur Steuerung des Systems	25
7.8	Initialisieren von Variablen und Klassen	26
7.9	Funktionen für die Übersichtlichkeit des Codes	27
7.10	Aufrufen von Funktionen und Variablen deklaration	28
7.11	Erstellen der Datei zum Speichern der Daten	29
7.12	Messen und Aufzeichnen der Entfernung	30
8.1	Importieren und Zuordnen von .csv Dateien	33
8.2	Umwandlung von Kugelkoordinaten zu kartesischen Koordinaten	35
8.3	Darstellung der Messwerte	35

1 Grundlagen Elektronik

1.1 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: 1.1). Wenn diese nun durch Diffusion bis zur Sperrsicht gelangen, driften die Ladungsträger entgegen der Sperrspannung in die jeweiligen Raumladungszonen, dies ist als Strom messbar. [8]

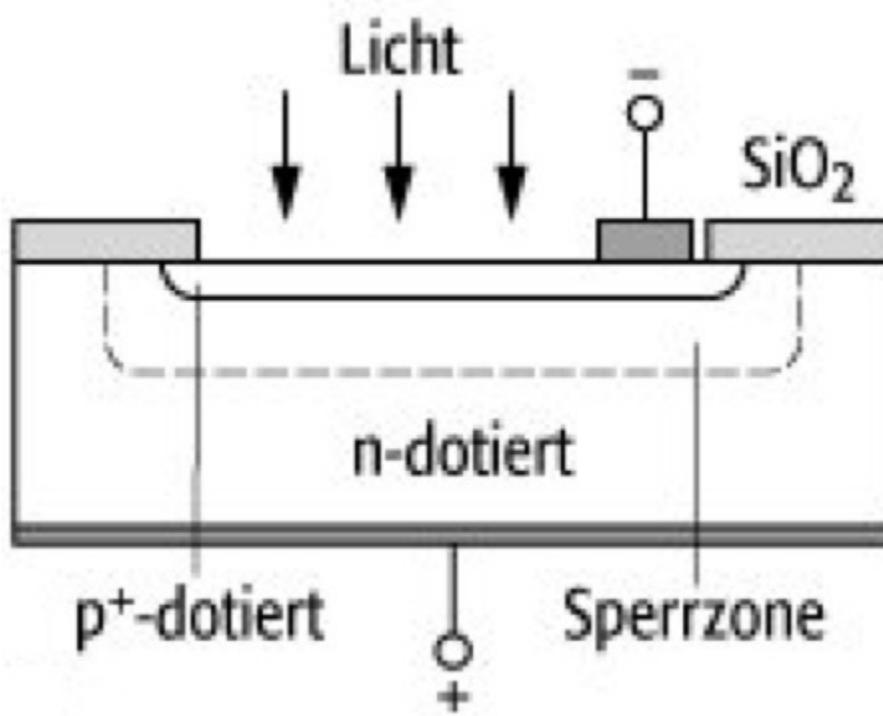


Abbildung 1.1: Schematischer Aufbau einer Photodiode [8] (p^+ starke p-Dotierung)

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.

1.1.1 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: 1.2).

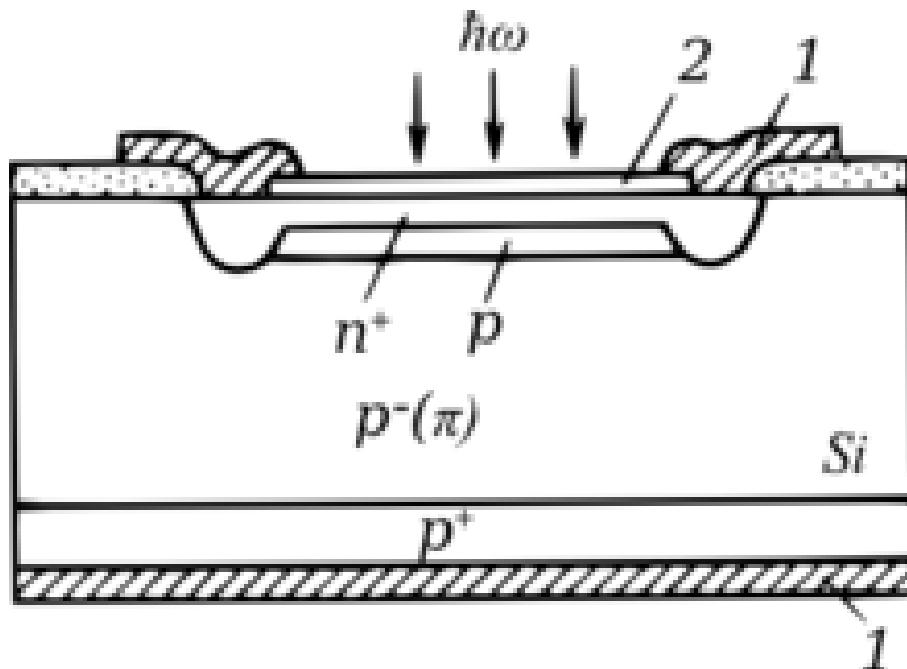


Abbildung 1.2: Schematischer Aufbau einer APD [1] (p^+ starke p-Dotierung, $p^-(\pi)$ schwache (intrinsische) p-Dotierung, n^+ starke n-Dotierung) (1 - Metallkontakte, 2 - Entspiegelung)

Wenn Photonen nun in die π 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. [5]

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. [7]

2 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.

2.1 Lichtlaufzeitmessung

2.1.1 Grundprinzip

Das Grundprinzip der Lichtlaufzeitmessung oder auch Time of Flight (ToF) (Abbildung: 2.1), bezieht sich auf die Zeit, welche ein ausgesandter Lichtimpuls benötigt bis er wieder am Sender eintrifft.

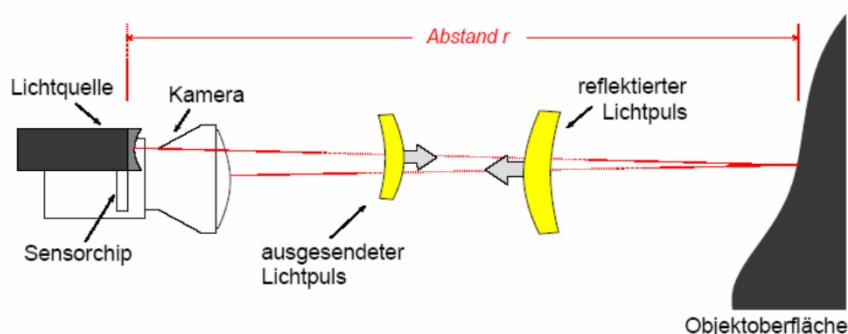


Abbildung 2.1: ToF Prinzip [3]

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. [4]

$$r = \frac{t_{diff}}{2} \cdot c \quad (2.1)$$

r = Abstand zum getroffenen Punkt [m]

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

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

2.1.2 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 $d_{min} = 1cm$
- Maximale Messentfernung $d_{max} = 1000m$
- Maximale Messfrequenz $f_{max} = 150kHz$

Formel 2.1 kann umgestellt werden, um die Zeiten zu errechnen, welche das Licht für die minimale und maximale Messentfernung benötigt.

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

$$t_{min} = \frac{0,01 \cdot 2}{299,79 \cdot 10^6} = 667,13 \cdot 10^{-9} \quad (2.2)$$

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

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.

2.2 Phasenverschiebung

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.

TODO: Phasenverschiebung

2.3 Triangulation

TODO: Triangulation

3 Grundlagen

3.1 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 Vorteile 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

4 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 geforderte 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 Punktes, bzw Zuordnung zu einer Wand, Decke

5 Mechanik

5.1 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.

5.2 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.

5.2.1 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 Punktwolke 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.

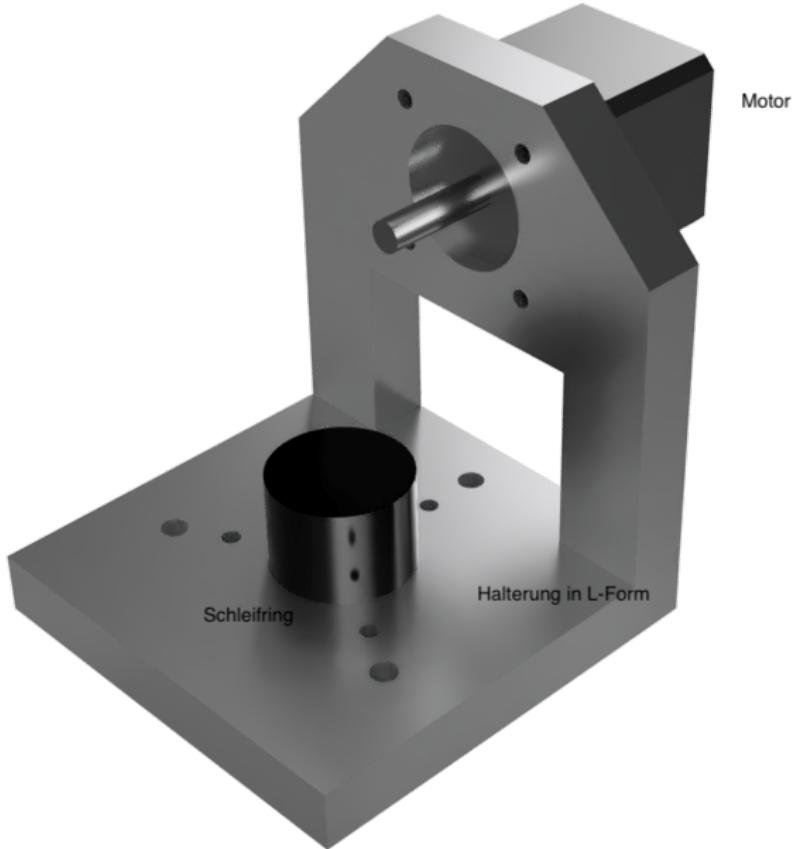


Abbildung 5.1: Oberer Aufbau der Mechanik

Der Motor welcher in Abbildung 5.1 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 [6]. 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 5.1 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.

5.2.2 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.

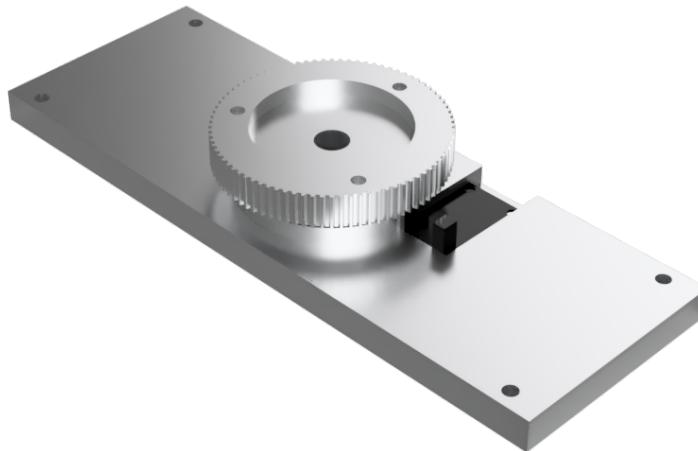


Abbildung 5.2: Basis der Mechanik

Um die Lagerung herzustellen wird ein großes Kugellager mit einem Innendurchmesser von $22mm$ in die Verbindungsplatte (Abbildung: 5.2) 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 [2] 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 [2]. 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.

5.2.3 Rahmen

Die dritte Baugruppe der Mechanik ist der Rahmen (Abbildung 5.3). 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° Drehen.

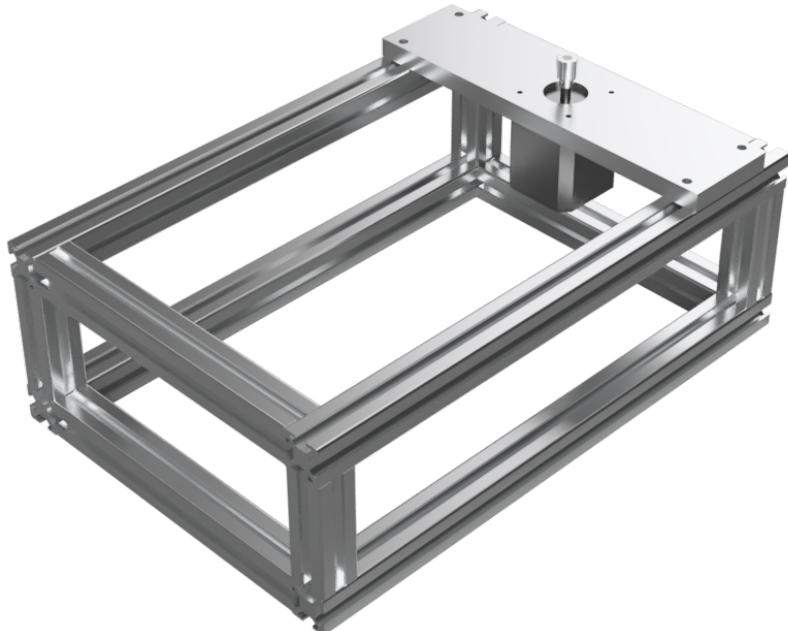


Abbildung 5.3: Rahmen

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:

$$i = \frac{z_a}{z_e} = \frac{84}{14} = 6 \quad (5.1)$$

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.

5.3 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.

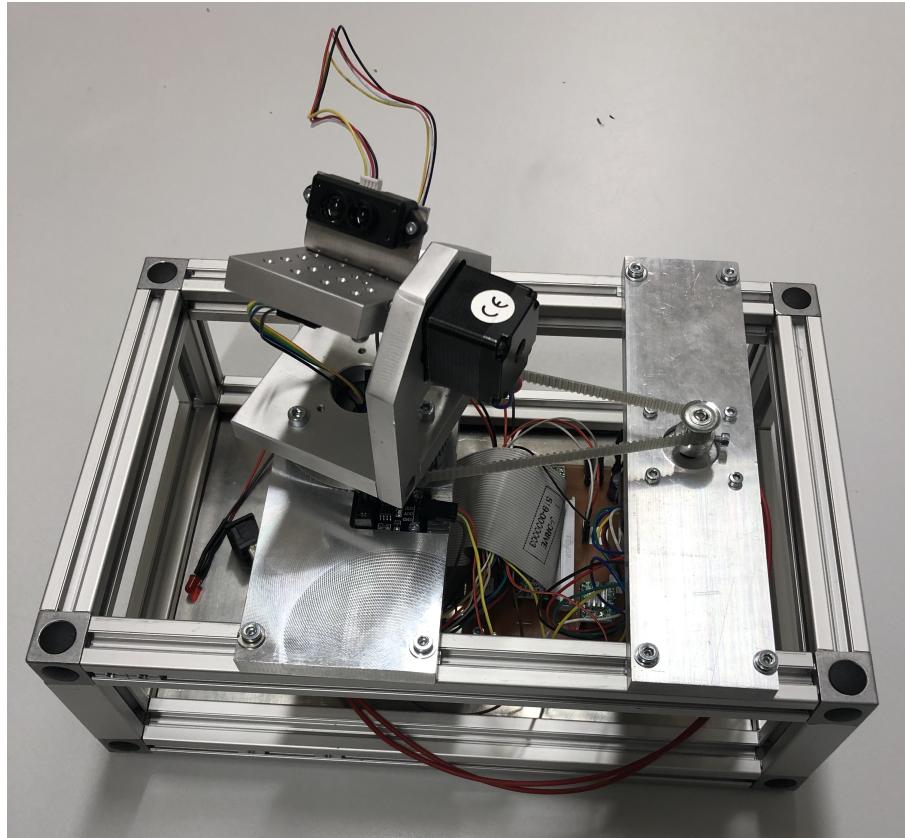


Abbildung 5.4: Mechanik mit montierter Elektronik

Nach Fertigstellung aller Einzelteile kann die Mechanik Zusammengebaut werden und die Elektronik eingebracht werden (Abbildung 5.4). 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 (*TODO: Section*) 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.

6 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 ist das Ausrichten des Sensors in zwei Achsen. Beinhaltet sind die zwei Schrittmotoren mit der damit verbundenen 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. Das Selbe 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 Outputs (GPIOs) und der Unterstützung aller benötigten Datenübertragungsprotokolle an. Zudem reicht die Rechenleistung für das Ansteuern aller Komponenten, sowie das Auswerten und Speichern der Messdaten aus.

6.1 Funktionseinheit Distanzbestimmung

6.1.1 TF MINI

Bei der Auswahl des Lidar Sensors spielen Preis, Verfügbarkeit, Messfrequenz, Auflösung, Genauigkeit und messbare Entfernung eine Rolle.

Als Sensor wird ein „TF MINI LiDAR“ von dem Hersteller „Seeed Studio“ verwendet. Dieser arbeitet mit auf dem Prinzip der Phasendifferenz wie in Kapitel ... beschrieben.

Der Arbeitsbereich liegt von 30 cm bis 1200 cm mit einer Auflösung von 1 cm. In dem Messbereich kleiner als 600 cm beträgt die Genauigkeit 1 %. Zwischen 600 cm und 1200 cm 2 %. Die Messfrequenz beträgt maximal 100 Hz.

Der Sensor arbeitet mit einer Versorgungsspannung von 4.5 V – 6 V bei einem durchschnittlichen Strom von 120 mA. Das Kommunikations-Interface ist UART mit einer Logikspannung von 3.3 V

Die Entscheidung viel auf diesen Sensor, da das Preis-Leistung Verhältnis sehr gut ist. Zudem reicht der messbare Bereich für den zuvor definierten Standardraum. Die Kommunikation über UART ist mit dem Raspberry Pi realisierbar und die 100 Hz reichen für Messungen, die nicht auf Geschwindigkeit ausgelegt sind, aus. Sensoren mit einer deutlich höheren Messfrequenz sind um vielfaches teurer.

Zudem ist der Sensor nur 42x15x16 mm groß, wodurch er sehr gut auf der dafür vorgesehenen Aluminiumhalterung montiert werden kann.



Abbildung 6.1: TFmini

6.1.2 VLX...

6.2 Funktionseinheit Ausrichtung des Sensors

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

Die horizontale Ausrichtung erfolgt zusätzlich über ein Getriebe mit einem Riemen zur Kraftübertragung. Das Übersetzungsverhältnis entspricht 1:6.

6.2.1 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 verhindert ungewolltes Verdrehen der Basis während der Messung. Dadurch ist das System fehlerresistenter auf äußere Einflüsse.

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 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.

6.2.2 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 bis zu 35 V und einem maximalen Phasenstrom von 2 A anzusteuern. Zudem sind mit dem Motortreiber Mikroschritte realisierbar. Dabei sind halb, viertel, achtel und sechzehntel Schritte möglich. Der maximale Ausgangsstrom ist über einen Potentiometer auf der Trägerplatine stufenlos einstellbar.

–ToDo: Zeichnung –

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 Versorgungs- spannung wird über ein 12 V Tischnetzteil bereitgestellt. Diese wird zusätzlich mit einem Stützkondensatoren geglättet, um Spannungsspitzen auszugleichen. Als Kontrolleinheit für den Motortreiber dient ein Raspberry Pi. Die Logikspannungsversorgung des Treibers wird mit einem 5 V Ausgang des Raspberry Pi's verbunden. Step, Dir, sowie MS1-MS3 werden mit GPIO's verbunden. Der Logikpegel an dem DIR-Pin legt die Bewegungsrichtung des Motors fest. Ein toggelndes Signal am Step Pin führt zur Rotation des Motors. Pro Steigende Flanke dreht sich der Motor um an den MS1-MS3 eingestellte Schrittweite weiter. MS1-MS3 dienen zum Einstellen der Schrittweite. Die Kombination der Logiklevel entscheidet dabei über die Schrittweite. Theoretisch wären somit $2^{hoch}3$ also 8 Kominationen möglich. Da jedoch nur fünf Einstellungen benötigt werden, sieht die Wahrheitstabelle Folgendermaßen aus:

Tabelle:

MS1	MS2	MS3	Auflösung
Low	Low	Low	Vollschrift
High	Low	Low	Halbschrift
Low	High	Low	viertel Schritt
High	High	Low	achtel Schritt
High	High	High	sechzehntel Schritt

Tabelle 6.1: Schrittweite

Um Beschädigungen an den Schrittmotoren zu vermeide, muss der maximale Strom durch die Spulen begrenzt werden. Dies kann man über einen Potentiometer auf der Oberseite des Treiberbausteines machen. Dafür wird die Referenzspannung zwischen dem Potentiometer und Masse gemessen. Mit der Formel : Current Limit = VRef *2 kann der maximale Strom ausgerechnet werden. Unterschiedliche Bauweisen des Treiberbausteines führen aber oftmals dazu, dass die Formel nur als grober Richtwert gewertet werden kann. Nach der groben Einstellung des Stromlimits sollte man den Strom bei aktivem Motor messen und gegebenfalls noch anpassen. Ändert man die Schrittweiten, ändert sich das Limit etwas...

Pro Motor wird ein Motortreiber auf der Platine verwendet.

6.3 Funktionseinheit Kalibrierung

6.3.1 Lichtschranke

Zum automatischen Positionieren der Basis wird eine Infrarot Lichtschranke verwendet. Diese befindet sich direkt unter dem Ritzel und detektiert das Durchlaufen des daran befestigten Kalibrierzapfens. Auf dem Lichtschrankenmodul ist LM393.. Das Modul benötigt eine Versorgungsspannung von 5V. Zudem wird der Ausgangspin 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 in der Lichtschranke und sie ist unterbrochen. Wechselt dieser Wert auf einen LOW Pegel, so ist die Lichtschranke nichtmehr unterbrochen.

6.3.2 Gyroensor

6.4 Platinen

6.5 Abwärtswandler

Da für einige Komponenten eine Spannung von 5V benötigt wird, muss die Versorgungsspannung von 12 V verkleinert werden. Dazu wird ein Abwärtswandler verwendet. Über ein Potentiometer auf der Oberseite des Moduls kann die Ausgangsspannung eingestellt werden. (MINI-360)

7 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.

7.1 Motor

Die erste Datei und Klasse beschäftigt sich mit der Ansteuerung der Schrittmotoren. Sie benötigt zwei extra Bibliotheken (Listing 7.1). 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 7.1: Bibliotheken der Motor Klasse

7.1.1 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 7.2) 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 7.2: Konstruktor der Motor Klasse

7.1.2 Bewegen des Motors

Die Motor Klasse besitzt zudem noch eine Funktion, mittels welcher sich der jeweilige Motor bewegen lässt (Listing 7.3). 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 7.3: Funktion zum Bewegen des Motors

7.2 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 7.4), 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 7.4: Bibliotheken der Lidar Klasse

7.2.1 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' *TODO: Referenz TFMINI* benötigt (Listing 7.5).

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 Inter-Integrated Circuit (I2C) (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 7.5: Konstruktor der Lidar Klasse

7.2.2 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())):
8                         Dist_L = self.ser.read()
9                         Dist_H = self.ser.read()
10                        self.dist = (ord(Dist_H) * 256) + (ord(Dist_L))
11                        for i in range (0,5):
12                            self.ser.read()
13                        self.recievedData = True
14                        break
15                 else:
16                     self.dist = self.tof.get_distance() # Entfernung in mm

```

16

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

Listing 7.6: Funktion um Distanz vom LIDAR Sensor zu erhalten

In Listing 7.6 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).

7.3 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 7.7: Bibliotheken zur Steuerung des Systems

Die Bibliotheken in Zeile 2 & 3 (Listing 7.7) 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*

7.3.1 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 7.8: Initialisieren von Variablen und Klassen

Zunächst werden die Pins für die verschiedenen auf der Platine vorgesehenen Funktionen definiert (Listing 7.8). 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 Serial Peripheral Interface (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 I2C zu verwenden.

7.3.2 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 7.9).

```

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 7.9: 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.

7.3.3 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 7.10: Aufrufen von Funktionen und Variablen deklaration

In Zeile 1 (Listing 7.10) 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. *TODO: FEHLER?!*
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.

7.3.4 Erstellen der Datei zum Speichern der Daten

Ein letzter Schritt ist noch nötig, bevor mit der Messung begonnen werden kann (Listing 7.11). 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 7.11: 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 7.1: 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 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. *TODO: Tabelle bsp Daten*

7.3.5 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 7.12: 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 ??, um zu Überwachen wie oft die Schleife schon durchlaufen wurde. Da in diesem Fall zwei "whileSSchleifen 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° erreicht. Im inneren dieser Schleife gibt es eine Besonderheit. Da sich die Kabel des oberen Aufbaus nur endlich oft drehen können bis diese verschleißen, muss das System nach jeden 360° 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° 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. *TODO: Hauptklasse*

8 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.

8.1 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 .. 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... extrahiert.

Anschließend werden in Zeile .. bis... 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 azimuth = data(:,3);
12 elevation = data(:,4)
```

Listing 8.1: Importieren und Zuordnen von .csv Dateien

8.2 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 θ und dem Azimutwinkel ϕ definiert wird.

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

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 π oder von 0 bis 2π . Für das Lidar System wird die zweite Definition verwendet.

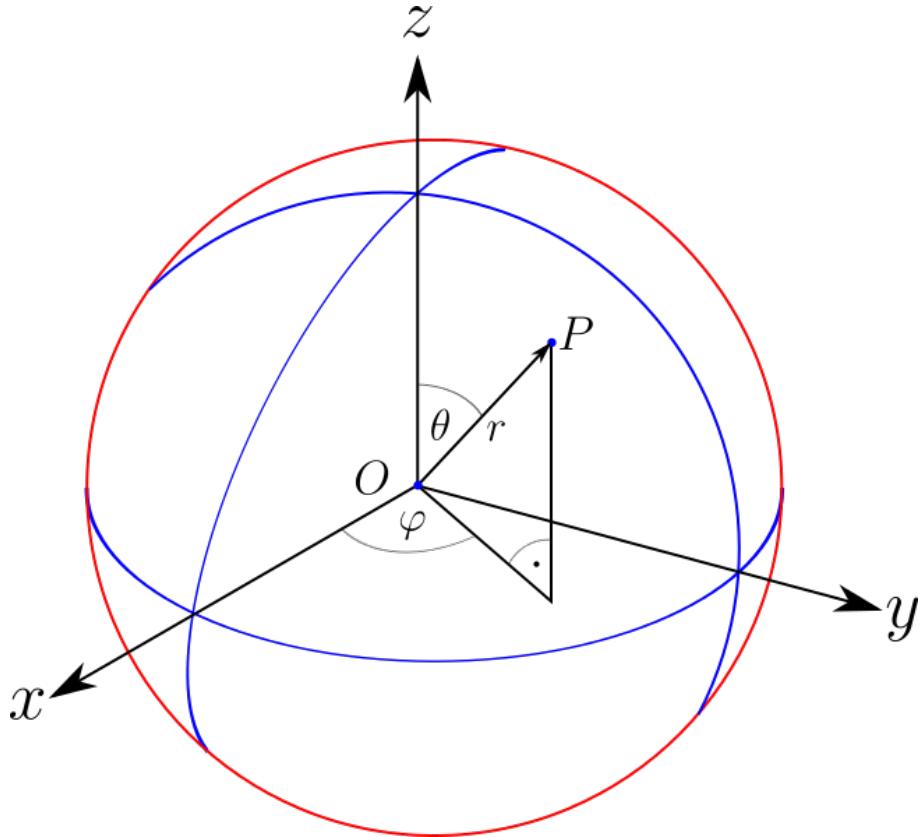


Abbildung 8.1: Kugelkoordinaten

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

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 experimentieller Basis und aufgrund der Größe des vorher definierten Standardraums festgelegt. Beim Lidar TF Mini werden nicht messbare Entfernung mit einer Entfernung von 34?? 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.

$$\begin{aligned}x &= r \cdot \cos(\theta) \cdot \cos(\phi) \\y &= r \cdot \cos(\theta) \cdot \sin(\phi) \\z &= r \cdot \sin(\theta)\end{aligned}\tag{8.1}$$

r = Abstand des Punktes zum Zentrum [m]

θ = Polarwinkel [$^{\circ}$]

ϕ = Azimutwinkel [$^{\circ}$]

```

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     y(i) = distance(i)*cos(deg2rad(elevation(i)))*sin(deg2rad(azimuth(i)));
5     z(i) = distance(i)*sin(deg2rad(elevation(i)));
6   else
7   end
8 end

```

Listing 8.2: Umwandlung von Kugelkoordinaten zu kartesischen Koordinaten

8.3 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
4 axis([-400 400 -400 400 0 240])
5 pbaspect([1 1 0.3])

```

Listing 8.3: 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 deiner 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 (Vgl Bild links und rechts oben - mitte)

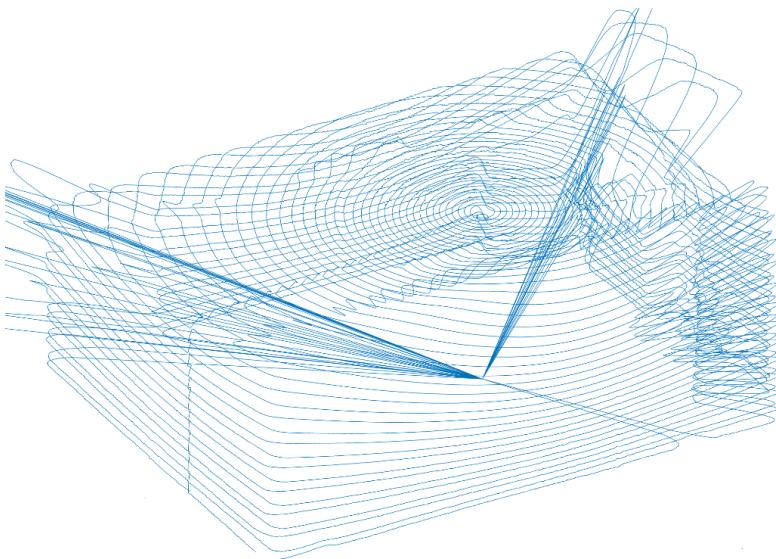


Abbildung 8.2: Darstellung mit Linien

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.

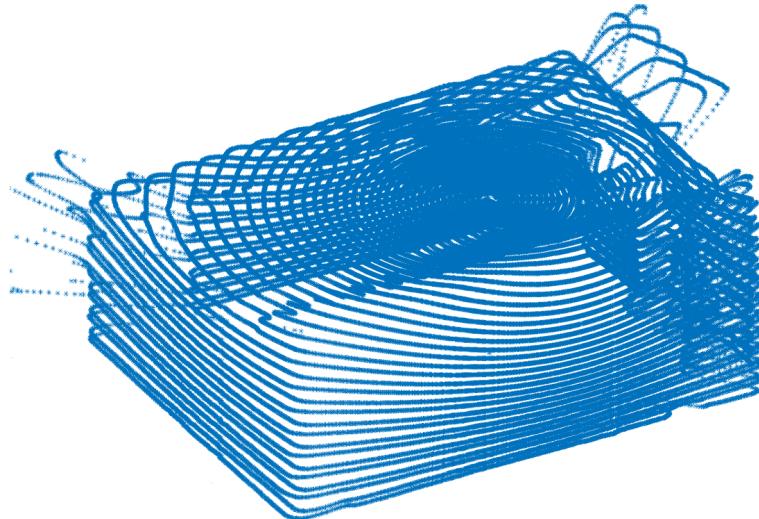


Abbildung 8.3: Darstellung mit Asterisken

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.

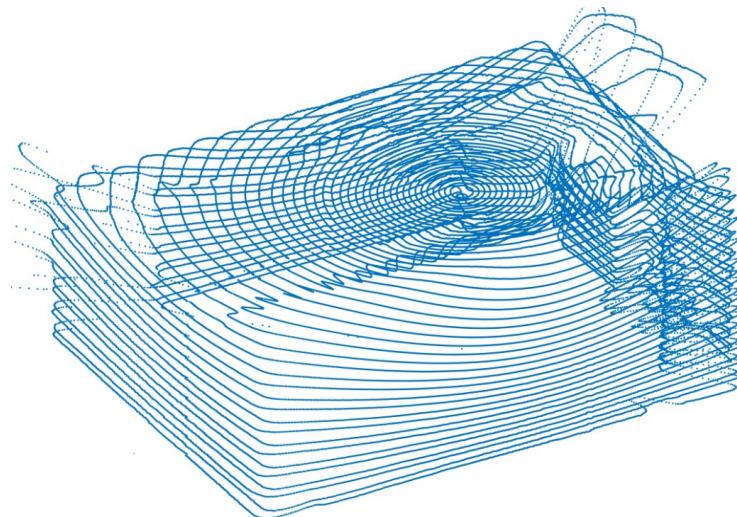


Abbildung 8.4: Darstellung mit Punkten

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 `axisEqual` kalibriert. Die ersten beiden Werte geben den x-Achsen in Zentimetern an. Der zweite und dritte den Wert der y-Achse und die letzten beiden 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.

9 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.

9.1 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 SSweet 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 Bild ... dargestellt. Alle weiteren Maße des Raumes können direkt in der Software abgerufen werden.

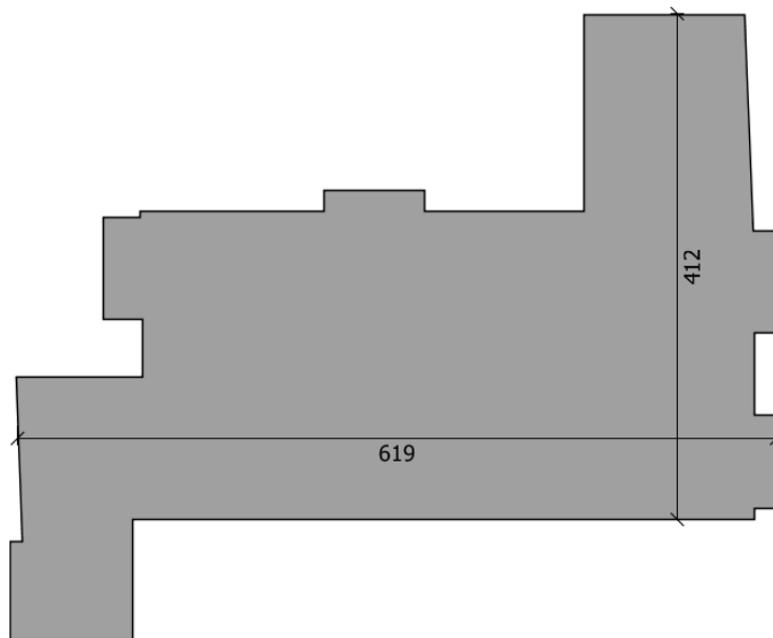


Abbildung 9.1: Grundriss des Testraums

Das Lidar-System wird im Raum aufgestellt, die Position ist annähernd mittig und wird zudem ausgemessen 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 ... stellen diese Schatten dar.

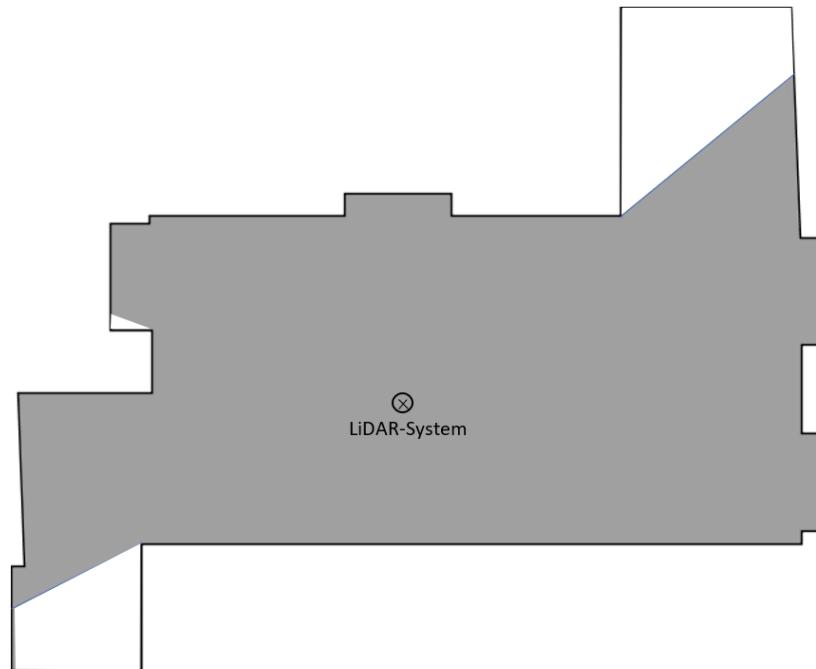


Abbildung 9.2: Grundriss des Testraums

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 eine Vogelperspektive des Raumes, bei dem der Grundriss auszumachen ist.

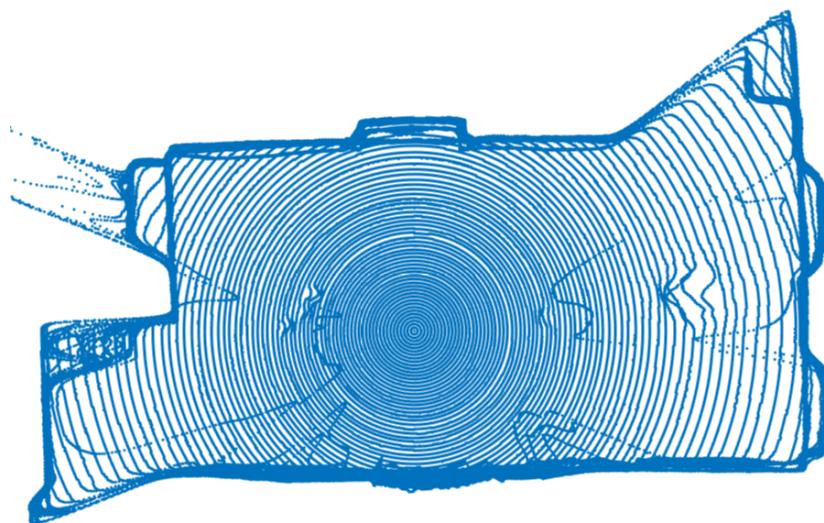


Abbildung 9.3: Vogelperspektive des Testraums

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

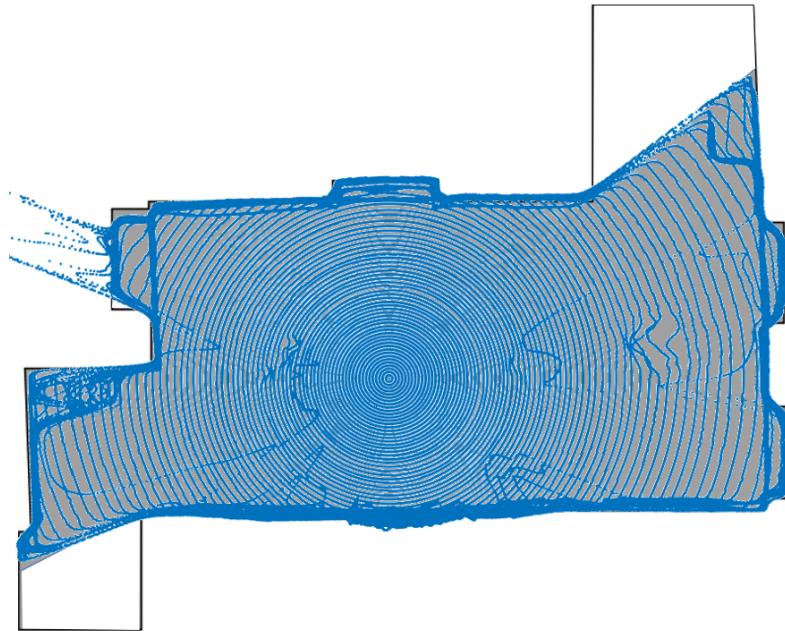


Abbildung 9.4: Grafischer Vergleich der Grundrisse

- Was kann man sehn? – nur 2 Linien nehmen
- Linien an Seite raus weil Glas
- Höhe wird überprüft Schrank, Bild usw

Messen mit Cursor

9.2 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 Schrittweite 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 immer der TF Mini Lidar verwendet.

9.2.1 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... berechnet werden.

$$d = \frac{360}{200 \cdot 8 \cdot 6} \cdot x \quad (9.1)$$

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.

$$d = \frac{360}{50 \cdot 4} \cdot x \quad (9.2)$$

Die Anzahl der Messpunkte lässt sich über die Anzahl der horizontalen Messpunkte * Anzahl der vertikalen Reihen berechnen.

Die Berechnung der Zeit hängt

	Auflösung gering	Auflösung mittel	Auflösung hoch
Auflösung horizontal [°]	1,2	0,15	0,15
Auflösung vertikal [°]	14,4	7,2	3,6
Anzahl Messpunkte	7524	120049	240099
Dauer [min]	4	?	?

Tabelle 9.1: Übersicht verschiedene Auflösungen

9.2.2 Vergleich der Ergebnisse

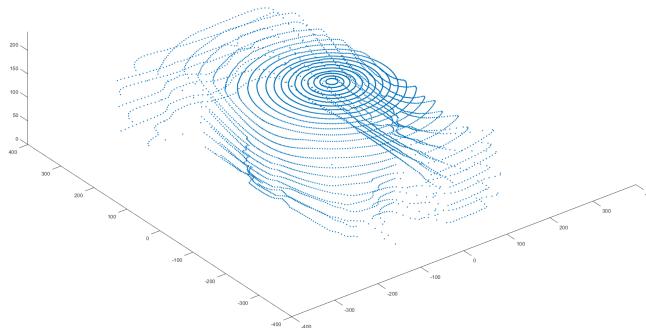


Abbildung 9.5: Beispielbild

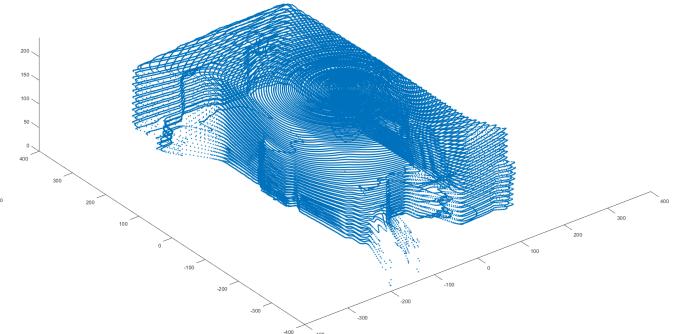


Abbildung 9.6: Beispielbild

9.3 Vergleich der Sensoren

Anhang

Literatur

- [1] Mikhantiev E. *The Structure of the silicon APD*. 2009. URL: <https://de.wikipedia.org/wiki/Datei:APD.png>.
- [2] Ulrich Fischer u. a. *Tabellenbuch Metall*. Europa Lehrmittel, 2011.
- [3] Nikolai Kutscher und Beate Mielke. *3D Kameras – basierend auf Lichtlaufzeitmessung*. 2005. URL: http://www.inf.fu-berlin.de/lehre/SS05/Autonome_Fahrzeuge/3d_Kameras.pdf.
- [4] 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.
- [5] Mamamatsu. *Characteristics and use of Si APD*. Mai 2004. URL: http://neutron.physics.ucsb.edu/docs/Avalanche_photodiodes_info.pdf.
- [6] *NEMA ICS 16*. National Electrical Manufacturers Association, 2001. URL: <https://www.nema.org/Standards/Pages/Motion-Position-Control-Motors-Controls-and-Feedback-Devices.aspx>.
- [7] PerkinElmer. *Avalanche photodiode - a user guide*. 2010. URL: http://www.perkinelmer.com/CMSResources/Images/44-6538APP_AvalanchePhotodiodesUsersGuide.pdf.
- [8] Spektrum. *Photodiode*. 1998. URL: <https://www.spektrum.de/lexikon/physik/photodiode/11179>.