TUHH
Technische Universität Hamburg-Harburg

Entwicklung eines Systems zur 3D-Rekonstruktion der dynamischen Umgebung mit einem rotierenden 2D-Laserscanner

D i p l o m a r b e i t
Mechatronik

- Peter Breuer -
25418

Erstprüfer:	Prof. Dr. Thomas Teufel
Zweitprüfer:	Prof. Dr. Jianwei Zhang
Betreuer:	Denis Klimentjew Ph. D. Houxiang Zhang

TAMS

UH

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die von mir am heutigen Tage eingereichte Diplomarbeit zum Thema

Entwicklung eines Systems zur 3D-Rekonstruktion der dynamischen Umgebung mit einem rotierenden 2D-Laserscanner

vollkommen selbständig verfasst und keine als die angegebenen Quellen und Hilfsmittel benutzt, sowie Zitate kenntlich gemacht habe.

Hamburg, den 31. März 2010

Kurzfassung

Roboter nehmen dem Menschen immer mehr Arbeiten ab und werden vor allem dort gerne eingesetzt, wo der Aufenthalt von Menschen nicht möglich oder gefährlich ist, wie in kontaminierten oder einsturzgefährdeten Umgebungen. Ein Beispiel für ein solches Einsatzgebiet das gerade von der Robotik erschlossen wird ist die Bergung nach Naturkatastrophen.

Eine genaue Kenntnis der Umgebung ist dabei von sehr großem Vorteil, der sich auf die Navigation, Pfadfindung und die Interaktion mit Gegenständen und Personen auswirkt. Da jedoch nur sehr selten ein exakter Umgebungsplan vorhanden und auch mit beweglichen Objekten oder Personen zu rechnen ist, sollte die Umgebung während des Einsatzes möglichst genau vermessen werden, um richtig reagieren zu können.

In dieser Arbeit war es die Aufgabe, ein System zu entwickeln, das auf einem mobilen Roboter montiert und mit dem die Umgebung dreidimensional vermessen werden kann. Dabei fand ein 2D-Laserscanner Verwendung, der mit einem in dieser Arbeit entwickelten Aufbau in eine rotatorische Bewegung versetzt wird und somit einen Blick für die gesamte Umgebung hat. Das System ist in der Lage, die Messdaten einzulesen und in eine dreidimensionale Punktewolke umzurechnen.

Die dreidimensionalen Datensätze können dann mit Hilfe eines PCs für weitere Berechnungen, wie zur Navigation oder Interaktion mit der Umgebung, verwendet werden.

Abstract

Robots increasingly adopt human operations and are constituted particularly in situations where the presence of human beings is impossible or dangerous as in contaminated environtments or places in danger of collapsing. An example of this application which is just being taken over by robotics is the rescue after natural catastrophes.

A precise knowledge of the environment is of very great advantage and makes navigation, path finding and interaction with objects and human beings much easier. Since an exact map of the area is very rarely available and as the presence of moving objects or human beings must be considered, the surrounding should be measured as accurately as possible, in order to be able to react correctly.

In this thesis, the task was to develop a system which is mounted on a mobile robot and by which the surrounding can be surveyed three-dimensionally. A two-dimensional laser range finder was used to perform a rotatory movement with a construction especially developed and designed in this work. With this one can get a view of the entire environment. The system is able to record the measuring information and convert it to a three-dimensional point cloud.

By using a computer for further calculations, this three-dimensional data then can be deployed for intelligent operations like navigation or interaction with the environment.

Inhaltsverzeichnis

Einleitung

Diese Diplomarbeit wurde von dem Autor, Student der Technischen Universität Hamburg-Harburg, am Institut für Informatik der Universität Hamburg in der TAMS[1]-Gruppe angefertigt.

Die Aufgabenstellung dieser Diplomarbeit bestand darin, ein System zu entwickeln, das es ermöglicht, mit Hilfe eines 2D-Laserscanners und einer mobilen Roboterplattform, dreidimensionale Bilder der Umgebung zu erstellen. Bei dem zu verwendenden Laserscanner handelte es sich um den URG-04LX von Hokuyo, bei der Roboterplattform um den Pioneer 2 DX von ActivMedia Robotics. Aufgrund der geringen Messreichweite des Scanners von maximal fünf Metern und der eingeschränkten Mobilität der Roboterplattform sollte sich der Einsatzbereich zunächst auf Umgebungen in geschlossenen Räumen beschränken. Eine weitere Anforderung an das System ist, dass es leicht auf andere Roboterplattformen portiert werden kann. Da außerdem für Messungen im Freien eine größere Messreichweite erforderlich ist, sollte der Hokuyo-Laserscanner durch einen leistungsstärkeren Laserscanner von IBEO ausgetauscht werden können. Der Vorteil eines eigens entworfenen Systems ist neben der Individualisierbarkeit auch der Preis, der gegenüber kommerziellen Systemen möglichst gering gehalten werden sollte.

Im Rahmen dieser Diplomarbeit wurde eine Rotationsplattform entwickelt, die den Laserscanner automatisiert bewegt, die Messdaten vorverarbeitet und an einen PC sendet, der die Hauptverarbeitung der Daten vornimmt. Als PC dient in dieser Arbeit ein Laptop mit einem Linux-Betriebssystem, der auf der Roboterplattform angebracht ist. Aufbauend auf diese Arbeit wird die Verarbeitung der Daten im Rahmen der Diplomarbeit von Gregor Michalicek vorgenommen. Er verwendet dazu sogenannte SLAM[2]-Algorithmen, mit deren Hilfe man trotz Mess- und Positionierungsfehler des Roboters und der Rotationsplattform dreidimensionale Karten erstellen kann.

Die Arbeit ist in vier Abschnitte gegliedert. Nachdem zuerst ein kurzer Überblick über vergleichbare Arbeiten gegeben wird, um den aktuellen Stand der Technik zu zeigen, beschäftigt sich der erste Abschnitt mit der Konstruktion der Rotationsplattform. Hier werden grundlegende Überlegungen erörtert, die in die Konstruktion eingeflossen sind. Außerdem findet man hier eine kurze Beschreibung der wichtigsten Komponenten und Erklärungen, warum diese für die Arbeit ausgewählt wurden.

Der zweite Abschnitt beschreibt die Software, die entwickelt wurde. Dabei teilt sich dieser Abschnitt in zwei Teile. Der erste beschreibt die embedded Software, also die Software, die auf dem Mikrocontrollerboard läuft und die Steuerung der Rotationsplattform übernimmt. Der zweite Teil gibt einen Überblick über das SLAM-Problem, das für die weitere Verwendung des Systems auf mobilen autonomen Robotern gelöst werden muss.

Im dritten Abschnitt sind einige Ergebnisse vorgestellt, die sich bei Reaktionszeitmessungen ergeben haben. Er dient dazu, zu zeigen, wie verlässlich die Messdaten sind und welche Grenzen dem System zugrunde liegen. Außerdem werden in diesem Abschnitt einige Messreihen grafisch dargestellt, um einen Eindruck über die Qualität des Messvorgangs zu geben.

Der vierte und letzte Abschnitt beschreibt, wie das System auf den leistungsstärkeren Laserscanner von IBEO umgerüstet wurde.

Im Anschluss befindet sich eine kurze Zusammenfassung der Arbeit und ein Ausblick auf zukünftige Erweiterungen und Verbesserungen.

1 Technical Aspects of Multimodal Systems (http://tams-www.informatik.uni-hamburg.de)
2 Simultaneous Localization and Mapping

Vergleichbare Arbeiten

Der Versuch mit Hilfe eines 2D-Laserscanners eine dreidimensionale Karte zu erstellen ist keine neue Idee. Es gibt mehrere kommerzielle Systeme und Arbeiten an verschiedenen Universitäten und Forschungseinrichtungen, die mit verschiedenen Ansätzen an das Problem herangehen und dieser Arbeit als Inspirationsquellen gedient haben. Aus den zahlreichen Beispielen seien hier drei charakteristische Arbeiten vorgestellt.

Der Hokuyo-Laserscanner URG-04LX wurde bisher nur selten in vergleichbaren Arbeiten verwendet. An der Universität Koblenz-Landau hat sich Christian Delis im Rahmen einer Studienarbeit mit der *„Entwicklung einer Rotationsplattform für den Hokuyo URG-04LX Laserscanner“* [14] beschäftigt. Die Messdaten sollten im Rahmen der Robocup Rescue-Mission zur Navigation benutzt werden, aus diesem Grund ist der Scanbereich und die höchste Punktintensität nach vorne ausgerichtet. Der Aufbau ist auf einem mobilen Roboter angebracht und der Laserscanner wird mit Hilfe eines Modellbauservos um die nach vorne gerichtete horizontale Achse rotiert. Da Modellbauservos jedoch nicht über 360° rotieren können, muss spätestens beim Erreichen des Anschlags die Rotationsrichtung geändert werden. Die Bewegung des Servos wird mit dem Synchronisationssignal des Laserscanners gesteuert, jedoch findet keinerlei Datenverarbeitung in der Elektronik statt, die für die Steuerung der Bewegung sorgt. Der Laserscanner wird direkt an einen PC angeschlossen. All das zusammen macht den Aufbau klein und nützlich zur Navigation, jedoch nicht zur Vermessung von Räumen.

Eine Arbeit an der Beijing University of Aeronautics and Astronautics mit dem Namen *„Fast Continuous 360 Degree Color 3D Laser Scanner“* [16] hat ebenfalls den Anspruch einen relativ preiswerten 3D Laserscanner zu realisieren. Allerdings werden dort die 2D Laserscanner LMS291 oder LMS200 der Firma SICK verwendet, die größer, schwerer und viel teurer sind als der URG-04LX. Die Rotation findet um die vertikale Achse statt, ist jedoch auf 360° beschränkt. Auch ist die Mobilität des Systems beschränkt, da es sehr schwer ist und einen PC mit PCI-Slot benötigt, um mit einer PCI-RS422-Erweiterungskarte die Messdaten aufnehmen zu können. Interessant an dieser Arbeit ist die Rotationsmechanik, die durch die Verwendung einer kommerziellen Rotationsplattform mit integriertem Schneckengetriebe und Servomotor besonders niedrig ausfällt, jedoch auch nicht preiswert zu haben ist.

Als drittes und letztes Beispiel sei die Forschungsarbeit des Fraunhofer Instituts in Sankt Augustin genannt. Die mit dem Titel *„6D SLAM – Preliminary Report on Closing the Loop in six Dimensions“* [17] überschriebene Arbeit beschäftigt sich hauptsächlich mit dem Zusammenführen verschiedener Messsysteme zu einem Gesamtbild. Wie in der davor beschriebenen Arbeit wird auch hier ein Laserscanner der Firma SICK verwendet. Die Rotation findet um eine horizontale Achse statt und ist auf 180° beschränkt, da, wie in der zuvor beschriebenen Arbeit, ein Servo für die Bewegung sorgt. Der Aufbau ist mit 7 *kg* recht schwer und die Datenverarbeitung wird von dem Hauptprozessor des Roboters übernommen.

Mit dem Wunsch, sämtliche Anforderungen zu erfüllen und inspiriert von den oben erwähnten Arbeiten wurde der Entwurf der Rotationsplattform in Angriff genommen.

I Konstruktion

1 Einleitung Abschnitt I

In diesem Abschnitt wird der technische Aufbau des in dieser Arbeit konstruierten Systems, dessen funktioneller Aufbau in Abbildung 1 dargestellt ist, beschrieben. Kernstück ist ein Mikrocontrollerboard, welches die Aufgabe hat, auf Steuersignale des PCs zu reagieren und eigene Steuersignale an den Laserscanner und den Schrittmotor zu senden. Außerdem empfängt es die Messdaten, vorverarbeitet diese und schickt sie an den PC. Das Synchronisationssignal des Laserscanners wird ebenfalls von dem Board empfangen und dient als Taktgeber für die Bewegung des Schrittmotors, der wiederum den Laserscanner dreht. Der ganze Aufbau ist so gestaltet, dass er ohne weiteres auf einer Roboterplattform montiert und der Laserscanner leicht ausgetauscht werden kann.

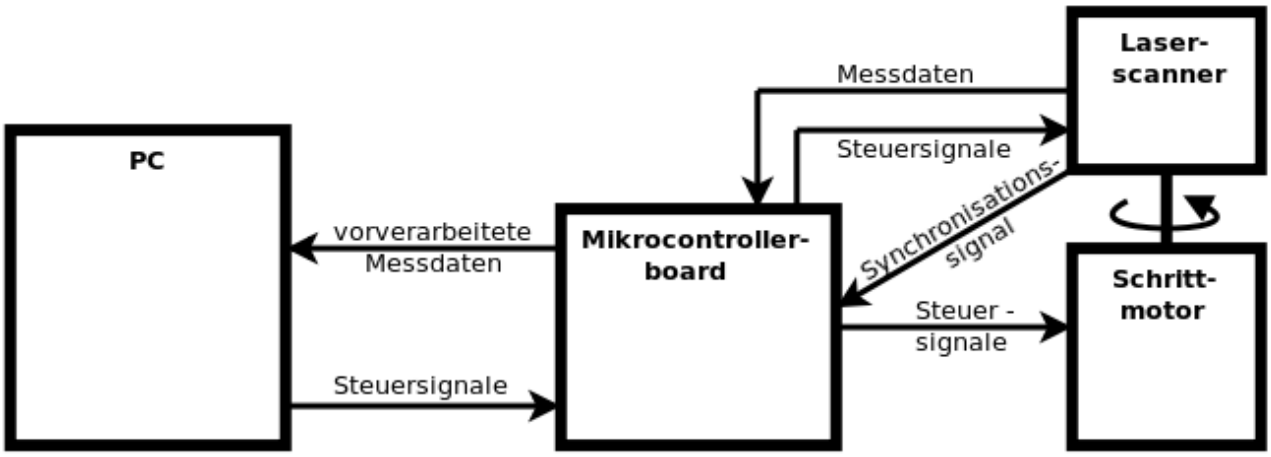


Abbildung 1: Schaltdiagramm des Messsystems

In die Konstruktion des Systems flossen viele Bedingungen ein, die berücksichtigt werden mussten. Während einige Teile schon vorhanden waren, mussten andere Komponenten ausgewählt und hinzugekauft werden. Nach ausführlicher Recherche wurde eine Liste mit Komponenten zusammengestellt, die den Anforderungen des Systems zu entsprechen schienen.

In Kapitel 2 werden sämtliche Komponenten kurz beschrieben und es wird erläutert welche Gründe zu deren Auswahl geführt haben. Kapitel 3 beschreibt die Vorüberlegungen und den Aufbau der Konstruktion. Der Abschnitt endet mit einer Zusammenfassung in Kapitel 4.

2 Beschreibung der Komponenten

2.1 Die Roboterplattform

Bei der Roboterplattform handelt es sich um den einachsigen Pioneer 2 DX von ActivMedia Robotics. Dieser wurde mit einer größeren Plattform erweitert, damit ein Notebook neben der Messvorrichtung platziert werden kann, siehe Abbildung 2. Der TAMS-Arbeitsgruppe stehen seit mehreren Jahren zwei Pioneer 2 DX und ein Pioneer 1 zur Verfügung, die zuverlässig ihre Aufgaben erfüllen.

Abbildung 2: Foto des Pioneer 2 DX

Als Stromversorgung dienen dem Pioneer 2 DX bis zu drei 12 *V*-Bleiakkus, die für eine durchschnittliche Laufzeit von über 24 Stunden sorgen. Die Roboterplattform besitzt zwei Räder, die unabhängig voneinander angetrieben werden können und ein Stützrad. Die Positionierung wird über Inkrementalgeber an den Motorachsen vorgenommen. Mit Hilfe von acht Ultraschallsensoren an der Vorderseite kann er Gegenstände, die bis zu fünf Meter entfernt sind detektieren und deren Abstand messen. Diese Funktion ist vor allem für die Kollisionsvermeidung sehr nützlich. Optional kann ein Zweibackengreifer an der Vorderseite montiert werden. Im Inneren bietet der Pioneer genügend Platz für die Aufnahme der Rotationselektronik.

2.2 Der Laserscanner

Bei dem verwendeten 2D Laserscanner handelt es sich um den URG-04LX. Er wird von der japanischen Firma Hokuyo Automatic CO., LTD hergestellt. Drei dieser Laserscanner stehen der TAMS-Gruppe zur Verfügung, einer davon ist in Abbildung 3 dargestellt. Er besitzt sehr kleine Abmessungen (50 *mm* x 50 *mm* x 70 *mm*) und ein geringes Gewicht von nur 160 *g* [2].

Abbildung 3: Foto des Laserscanners URG-04LX

Der URG04-LX (Wellenlänge λ=758 *nm*) arbeitet nach dem Prinzip der Phasenverschiebung. Dabei wird ein modulierter Infrarot-Laserstrahl ausgesandt, der an der Oberfläche des Körpers, dessen Entfernung bestimmt werden soll, reflektiert wird. Nun wird die Phasendifferenz des ausgesandten und des reflektierten Laserstrahls ermittelt, aus der die Entfernung des Körpers berechnet werden kann. Der Messbereich des URG-04LX liegt zwischen 20 *mm* und 4000 *mm*, wie aus dessen Datenblatt [2] ersichtlich ist. Diesem kann man auch entnehmen, dass die Messgenauigkeit bei einer Entfernung bis zu einem Meter bei ±10 *mm* Abweichung liegt und darüber bei etwa 1% der gemessenen Distanz.

Allerdings ist in der Praxis die maximale Messreichweite sehr viel größer und es hängt von dem Protokoll ab, das zur Kommunikation mit dem URG04-LX verwendet wird, wie weit man messen kann. Mit dem sogenannten SCIP1.1-Protokoll kann eine Reichweite von 4095 *mm* erreicht werden, mit dem SCIP2.0-Protokoll bis zu 5600 *mm*. Genaueres ist in der Beschreibung des SCIP2.0-Protokolls [1] zu finden.

Die Messgenauigkeit ist sehr von der Oberflächenbeschaffenheit des Gegenstandes abhängig, dessen Abstand bestimmt werden soll [18]. So werden zum Beispiel Materialien mit einer hellen oder stark reflektierenden Oberfläche auf eine kürzere Entfernung gemessen, als sie sich tatsächlich befinden. Weiche Materialien wie Schaumstoffe oder Schwämme ergeben wiederum größere Messwerte als sie haben sollten. Gegenstände die weniger als 20 *mm* von dem Scanner entfernt sind, können nicht vermessen werden, da in den Protokollen des URG04-LX alle Messwerte kleiner als 20 einer Fehlermeldung entsprechen. Aus diesem Grund liefern Gegenstände, deren Abstand zum Scanner größer als die maximale Reichweite ist, auch einen Messwert kleiner als 20 *mm*. Ebenso verhält es sich mit Materialien, die den Laserstrahl nur schwach reflektieren. Auch wenn eine Oberfläche um 45° oder mehr zum Laserstrahl gedreht ist, kann der Scanner Schwierigkeiten bei der Messung haben und einen Fehlerwert liefern.

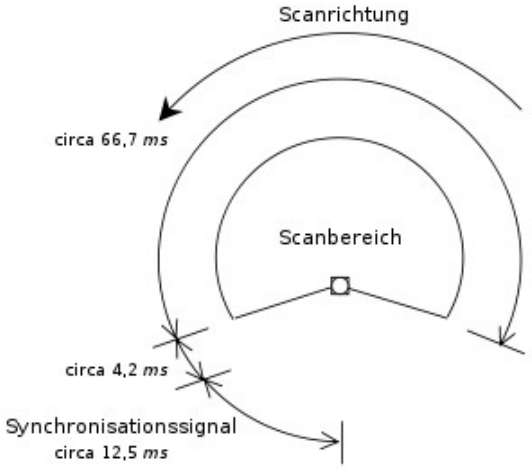


Abbildung 4: Scanverlauf des URG-04LX [2]

Die Messung einer Ebene funktioniert mit Hilfe eines rotierenden Spiegels. Durch diesen wird der Laserstrahl des URG04-LX abgelenkt, so dass in einem Bereich, der etwa 240° aufspannt, gemessen werden kann. Die Auflösung in der Ebene liegt bei ungefähr 0,35° (360° / 1024).

Der komplette Scan einer Ebene dauert etwa 100 *ms*. Abbildung 4 zeigt den Scanverlauf des URG-04LX. Nachdem der Scan gestartet wurde, läuft die Messung circa 66,7 *ms*. Nach einer

kurzen Zeitspanne von circa 4,2 *ms* beginnt ein Synchronisierungssignal mit einer Länge von ungefähr 12,5 *ms* auf einer eigenen Leitung. Bei dem Ausgang des Synchronisationssignal handelt es sich um einen sogenannten Open Collector-Ausgang. Das bedeutet, dass ein Pull-up-Widerstand zwischen der 5 *V*-Versorgungsspannung und der Signalleitung anliegen muss, damit das Signal von außen abgreifbar ist. In den Scanner ist dieser Widerstand nicht integriert. Daher muss, wie in Abbildung 5 dargestellt, ein 1 *kΩ* Widerstand am Ausgang des Scanners angebracht werden.

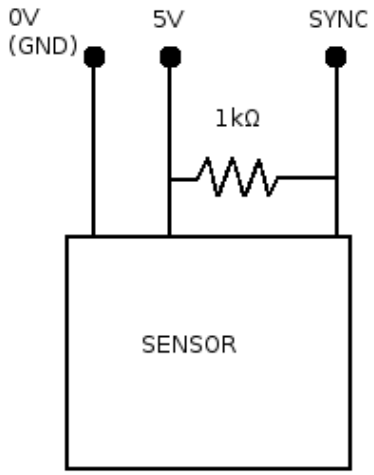


Abbildung 5: Open Collector Schaltung [14]

Der Datenversand kann über eine serielle Schnittstelle und über USB erfolgen. Die serielle Schnittstelle unterstützt folgende Baudraten: 19,2 *kbps*, 57,6 *kbps*, 115,2 *kbps*, 250 *kbps*, 500 *kbps* und 750 *kbps*. Mit dem USB-Anschluss kann man bis zu 12 *Mbps* erreichen. Diese Arbeit verwendet jedoch ausschließlich den seriellen Anschluss, wobei die standardmäßige Baudrate von 19,2 *kbps* für diese Arbeit auf 500 *kbps* erhöht wurde.

Die Spannungsversorgung von 5 *V* muss immer separat angeschlossen werden, eine Versorgung über den USB-Anschluss reicht nicht aus. Die durchschnittliche Stromaufnahme liegt laut Datenblatt [2] bei 500 *mA* oder weniger, der Spitzenwert bei 800 *mA*.

Der Hokuyo URG04-LX kann mit zwei verschiedenen Reichweiten betrieben werden, 4095 *mm* und 5600 *mm*. Die Messdaten werden bei der Verwendung der kürzeren Reichweite mit zwei, bei der größeren Reichweite mit drei Byte verschlüsselt. Dabei wird der Messwert in zwei, beziehungsweise drei Hexadezimalzahlen umgewandelt und diese als entsprechende ASCII-Zeichen versendet. Näheres zur Funktionsweise kann der SCIP2.0-Protokoll Beschreibung [1] entnommen werden.

2.3 Das Mikrocontrollerboard

Nach langer Recherche wurde das Mikrocontrollerboard SBC6000X von EmbestInfo & Tech CO., LTD für diese Anwendung gewählt und erworben (Abbildung 6). Es verfügt über einen leistungsstarken AT91SAM9261S-Prozessor von Atmel und eine Reihe von Anschlüssen zu einem verhältnismäßig günstigen Preis.

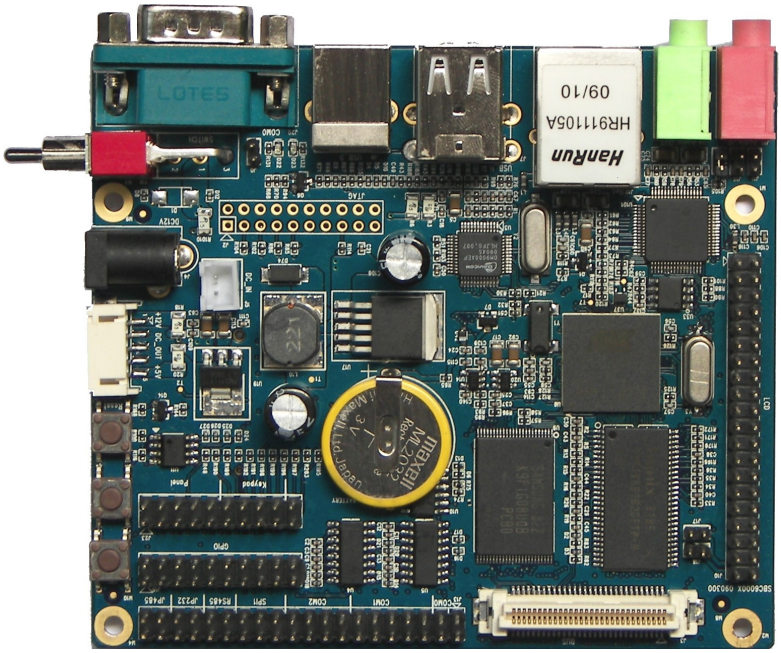


Abbildung 6: Das Mikrocontrollerboard SBC6000X [5]

Dem Datenblatt [5] sind folgende Eigenschaften zu entnehmen:

- Abmessungen: 106,4 *mm* x 96 *mm*
- 64 *Mbyte* SDRAM
- 128 *Mbyte* Nand Flash
- ein 10/100 *Mbit* Ethernet Anschluss (RJ45)
- drei serielle Anschlüsse (COM0 & COM1: RS232 / TTL, COM2: RS232 / TTL / RS485)
- zwei USB-Host Anschlüsse
- ein USB-Device Anschluss
- 16 GPIOs[3]
- SD-Karten Steckplatz
- Audio Ein- und Ausgang
- ein LCD-Anschluss
- ein 4 x 4 Tastaturanschluss

Ein embedded Linux mit der Kernel-Version 2.6.24 ist vorinstalliert. Laut Herstellerangaben ist das Board auch Windows CE-fähig.

Die für diese Arbeit relevanten Anschlüsse sind die drei seriellen Schnittstellen und der USB-

3 GPIO - General Purpose Input/Output – frei programmierbarer Eingang/Ausgang

Device Anschluss. Die serielle Schnittstelle COM0 ist als D-Sub DE-9-Stecker ausgeführt und so konfiguriert, dass man darüber auf das Betriebssystem zugreifen kann. So kann man auch ohne LCD und Tastatur auf dem Board Befehle ausführen. COM0 steht in dem System als Debugging-Schnittstelle weiterhin zur Verfügung, die Datenübermittlung zum PC findet allerdings über den USB-Device-Anschluss statt. Da jedoch hinter der USB-Device-Schnittstelle ein USB-to-Serial-Converter sitzt, kann die volle Geschwindigkeit von USB 2.0 nicht genutzt werden, sondern nur die langsameren für serielle Schnittstellen üblichen Baudraten. Die beiden übrigen seriellen Schnittstellen, COM1 und COM2, dienen zur Kommunikation mit dem Laserscanner und dem Schrittmotor.

Von den 16 GPIOs wird in dieser Arbeit nur einer benötigt, der das Synchronisationssignal des Laserscanners erkennen soll. Für diese Aufgabe kann man die GPIOs einzeln in einen Interrupt-Modus versetzen.

Das Board wurde auch deshalb ausgewählt, weil es über eine Ethernetschnittstelle verfügt und daher die Möglichkeit der Erweiterung des Systems auf den IBEO-Laserscanner bietet, siehe Abschnitt IV.

2.4 Der Schrittmotor

Wie später in Kapitel I.3.1 beschrieben wird, soll der Laserscanner eine durchgehende rotatorische Bewegung durchführen. Dafür wurde die Verwendung eines Schrittmotors gewählt, der gegenüber einem Servo, wie er aus dem Modellbau bekannt ist, über keinen Anschlag verfügt und daher eine durchgehende Rotation zulässt. Außerdem lassen sich Schrittmotoren mit dem sogenannten Microstepping sehr genau im Bereich von wenigen Zehntel Grad ansteuern.

Mit dem PD3-110-42-232 von Trinamic Motion Control GmbH & Co. KG, siehe Abbildung 7, steht ein Schrittmotor mit passender Ansteuerungseinheit zur Verfügung, die sich über eine serielle RS232-Schnittstelle ansprechen und steuern lässt. Dank der Programmiersprache TMCL[4] von Trinamic kann man Befehle direkt an den Motor schicken oder sogar Skripte ausführen lassen, die in dem Speicher der Ansteuerungseinheit abgelegt werden und bei Bedarf ausgeführt werden können. Der Aufbau der TMCL ist in der Bedienungsanleitung [4] ausführlich beschrieben.

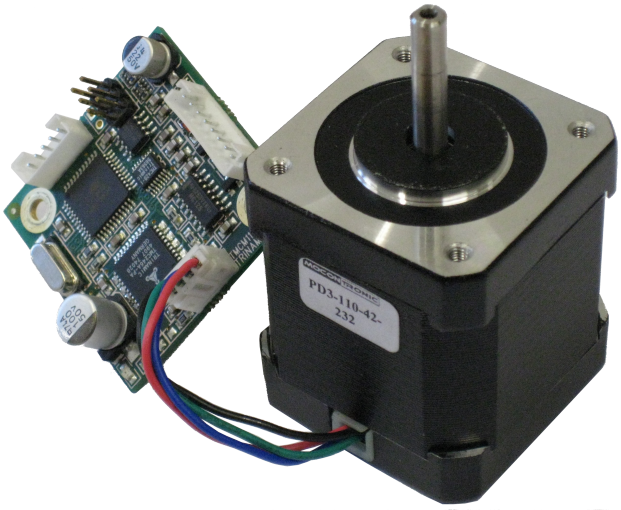


Abbildung 7: Foto des Schrittmotors mit Steuereinheit

4 Trinamic Motion Control Language

Bei der Auswahl des Schrittmotors war das Drehmoment ein entscheidendes Kriterium. Um das benötigte Motordrehmoment zu berechnen, muss man das Trägheitsmoment des rotierenden Aufbaus kennen, wofür man wiederum dessen ungefähres Gewicht und Form benötigt. Da zu diesem Zeitpunkt lediglich eine grobe Vorstellung der Abmessungen und des Gewichts vorhanden war, wurden die Werte überschlagen und eine großzügige Sicherheit hinzu addiert.

Die Formel für das Drehmoment [13] lautet:

$$M = J\,\ddot{\varphi} \tag{1}$$

Dabei entspricht J dem Trägheitsmoment und $\ddot{\varphi}$ der Winkelbeschleunigung.
Zur Berechnung des Trägheitsmoments des Aufbaus wurde folgende Formel verwendet:

$$J = ml^2 \tag{2}$$

Es handelt sich um eine Näherung, die nur den Steineranteil berücksichtigt, der in diesem Fall jedoch ausschlaggebend ist. m beschreibt die Masse des Aufbaus und l die Entfernung des Massenschwerpunkts von der Drehachse. Nimmt man für die Masse großzügig 1 *kg* und für die Entfernung 7 *cm* an, so ergibt sich ein Massenträgheitsmoment von J=49 *kgcm²*.

Um die Winkelbeschleunigung zu ermitteln, wurde angenommen, dass die Zeit, die für die Verstellung des Motors um 1° (~0,018 im Bogenmaß) benötigt wird, maximal 30 *ms* betragen darf. Dieser Wert ergibt sich aus der Scanpause des Laserscanners, siehe Kapitel I.2.2. Somit beträgt die mittlere Winkelgeschwindigkeit:

$$\dot{\varphi} = \frac{0{,}018}{0{,}03\,s} = 0{,}6\,\frac{1}{s} \tag{3}$$

Geht man von einer konstanten Winkelbeschleunigung aus, so bedeutet das, dass innerhalb von 15 *ms* die doppelte Winkelgeschwindigkeit, also 1,2 *1/s,* erreicht werden muss, damit die 30 *ms* nicht überschritten werden. Die Winkelbeschleunigung berechnet sich daraus zu:

$$\ddot{\varphi} = \frac{1{,}2\,\frac{1}{s}}{0{,}015\,s} = 80\,\frac{1}{s^2} \tag{4}$$

Eingesetzt in Formel 1 ergibt sich das benötigte Motordrehmoment.

$$M = J\,\ddot{\varphi} = 39{,}2\,Ncm \tag{5}$$

Der PD3-110-42-232 bietet daher mit einem Motordrehmoment von 49 *Ncm* einen ausreichend starken Motor.

Wichtig für die Wahl des Motors war auch die bereitgestellte Kommunikationsschnittstelle, da das Mikrocontrollerboard dementsprechend ausgerüstet sein musste. Mit der RS232-Schnittstelle des PD3-110-42-232 ist eine allgemein verbreitete Schnittstelle vorhanden, die auf den meisten Mikrocontrollerboards in mehrfacher Ausführung vorhanden ist.

2.5 Der Schleifring

Um eine durchgehende Drehbewegung garantieren zu können und um ein Aufwickeln der Daten- und Stromversorgungskabel zu vermeiden, kann auf die Verwendung eines Schleifrings nicht verzichtet werden. Bei der Auswahl des Schleifrings istneben der Anzahl der durchgeführten Leitungen zu beachten, dass die Qualität der Übertragung für die Anwendung garantiert wird. Der Schleifring LPT012-2405 der chinesischen Firma Jinpat Electronics Co., Ltd erfüllt laut Herstellerangaben diese Voraussetzung und ist in Abbildung 8 dargestellt. Er wurde gewählt, weil er ein hervorragendes Preis-Leistungs-Verhältnis besitzt und für die spätere Erweiterung des Systems ebenfalls ausgestattet ist. Mit den 24 Übertragungsleitungen kann man sowohl den Hokuyo- als auch den IBEO-Laserscanner nutzen und es stehen noch weitere Leitungen zur Verfügung, die für zusätzliche Geräte verwendet werden können. Die maximale Stromstärke, die der Schleifring pro Leitung übertragen kann, ist mit 5 *A* wesentlich höher als die Stromaufnahme der Laserscanner.

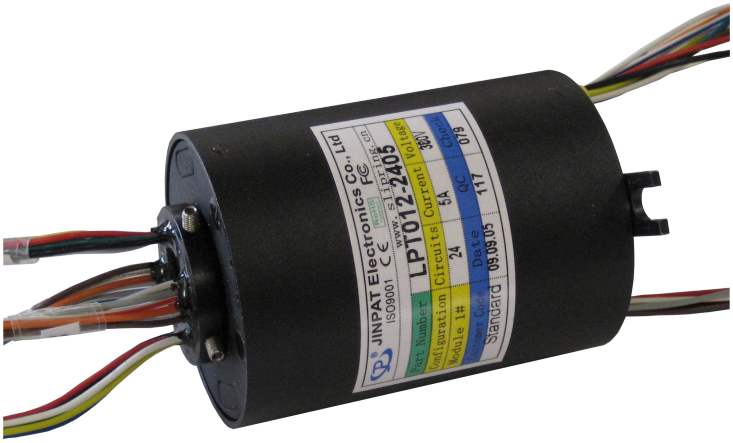


Abbildung 8: Foto des Schleifrings

2.6 Die Kupplung

Um Spannungen und Verkantungen im System zu vermeiden und einen ruhigen Bewegungsablauf zu garantieren, wurde eine Kupplung eingebaut. Die Kupplung Controlflex Compact der Firma Schmidt-Kupplung GmbH soll laut Herstellerangaben den axialen, radialen und winkligen Versatz in der Konstruktion ausgleichen und dabei drehstarr sein. Sie wurde für die Arbeit von Schmidt-Kupplung GmbH zur Verfügung gestellt.

Abbildung 9: Foto der Kupplung

2.7 Das Wälzlager

Bei dem Wälzlager handelt es sich um das käfiglose axiale Nadellager AXK6085. Es wurde eingebaut, um die Rotation bei möglichst niedrigem Widerstand zu erreichen. Da bei dem System relativ geringe Lagerkräfte auftreten und zwar fast ausschließlich in axiale Richtung, konnte bei der Wahl des Wälzlagers nahezu frei entschieden werden. Die sehr geringe Bauhöhe käfigloser Nadellager war entscheidend bei der Auswahl. Der recht große Durchmesser des AXK6085 sollte eine große Auflagerfläche der Rotationsplattform ermöglichen und für mehr Stabilität sorgen.

Abbildung 10: Foto des Nadellagers

3 Beschreibung der Konstruktion

3.1 Bewegung des Laserscanners

Zunächst musste geklärt werden, welche Bewegung der Laserscanner durchführen soll, um optimale Ergebnisse zu erzielen.

Dabei steht einer durchgehenden eine teilweise Rotation entgegen. Bei einer teilweisen Rotation wird meist die Drehrichtung des Scanners nach jeweils 180° geändert. Dies reicht für eine Vermessung der kompletten Umgebung aus, sofern der Scanner so angebracht ist, dass er gleichzeitig in zwei entgegengesetzte Richtungen scannen kann. Es führt jedoch auch dazu, dass die meisten Punkte in unregelmäßigen Abständen abgetastet werden. So wird zum Beispiel ein Punkt, der kurz vor Änderung der Drehrichtung abgetastet wurde, direkt darauf wieder abgetastet. Die darauffolgende Abtastung findet dann aber erst nach einer sehr viel längeren Zeit statt, da der Scanner einen langen Weg zurücklegen muss bis er wieder am Ausgangspunkt angelangt ist.

Eine durchgehende Rotation sorgt für eine gleichmäßige Abtastung aller Punkte. Man kann daher auch bei einer sich verändernden Umgebung davon ausgehen, dass das Bild in allen Punkten gleich aktuell ist. Der Nachteil der durchgehenden Rotation ist, dass zusätzliche Komponenten benötigt werden. Ein Schleifring muss dafür sorgen, dass sich keine Kabel aufwickeln können. Auch die Verwendung von relativ preiswerten Servos ist mit der durchgehenden Rotation nicht möglich, da Servos im allgemeinen über einen Anschlag verfügen und nicht über 360° drehen können. Die Verwendung eines Schrittmotors hat neben dem Vorteil, dass eine durchgehende Rotation ermöglicht wird, den weiteren Vorteil, dass es passende Ansteuerungseinheiten gibt, die eine sehr genaue Positionierung und leichte Kommunikation ermöglichen.

Ein wichtiger Punkt der für die durchgehende Rotation spricht, ist im Ausblick erwähnt. Man kann ein System so erweitern, dass die Rotation mit einer konstanten Geschwindigkeit stattfindet und die Messpunkte eines Scans nicht in einer Ebene, sondern auf einer Spirale liegen. Das würde eine schnellere Bewegung ermöglichen und aktuellere Messungen liefern. In dieser Arbeit jedoch stoppt

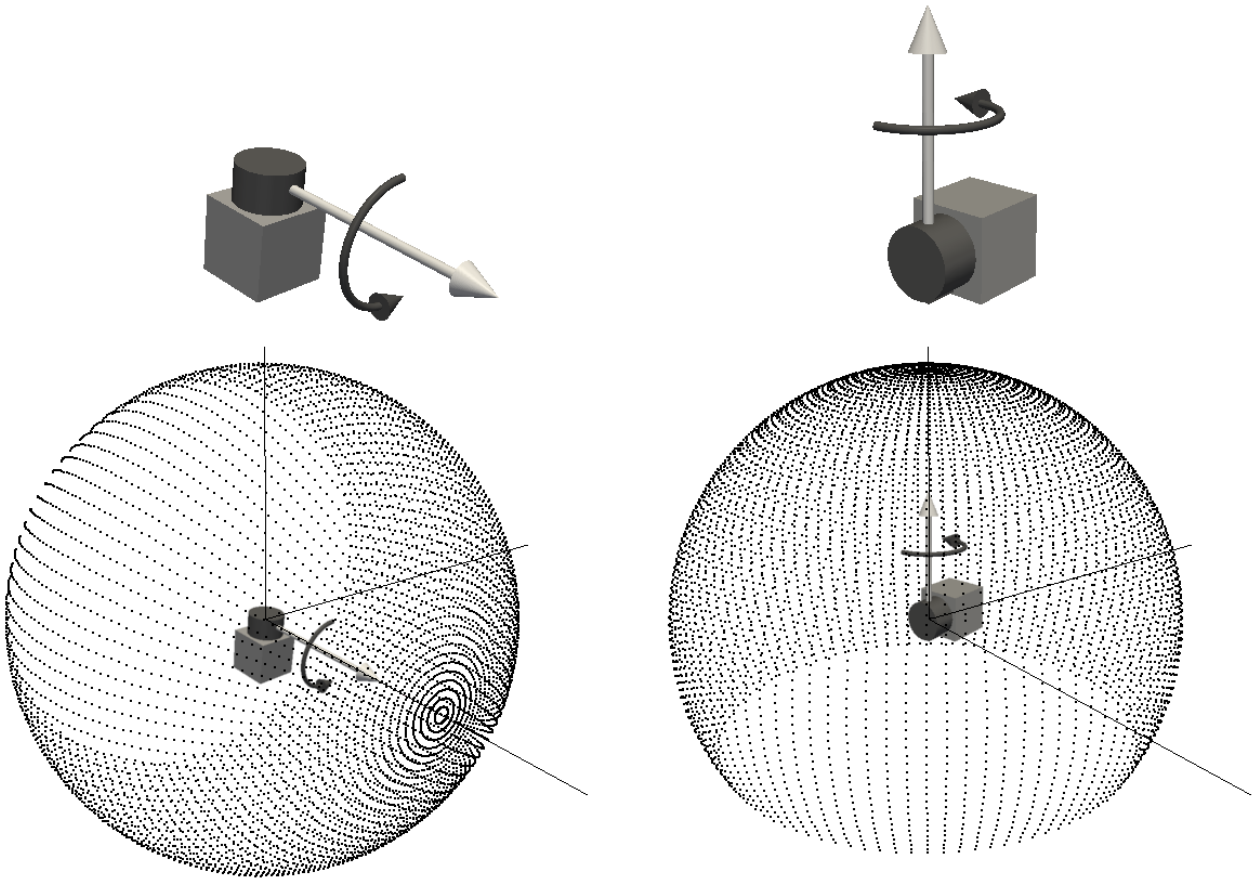
der Motor während des Messvorgangs und die Bewegung um eine bestimmte Gradzahl findet zwischen zwei Scans statt.

Nach dem Abwägen der Vor- und Nachteile wurde die durchgehende Rotation gewählt und die benötigten Komponenten besorgt.

3.2 Ausrichtung des Laserscanners

Die Ausrichtung des Laserscanners ist von entscheidender Bedeutung. Bedenkt man, dass der Laserscanner einen toten Winkel von 120° besitzt, so liegt es nahe ihn so anzubringen, dass sich die Roboterplattform während des gesamten Messvorgangs in diesem befindet. So müssen keine Messpunkte verworfen werden, die keine Informationen über die Umgebung liefern, sondern nur den Roboter zeigen.

Damit stehen lediglich zwei sinnvolle Ausrichtungen zur Auswahl. Bei der einen ist der Scanner vor dem Roboter angebracht, mit dem toten Winkel im Rücken. Die Rotation findet um die nach vorne gerichtete Achse statt, die höchste Punktedichte liegt direkt vor dem Roboter. Diese Ausrichtung ist in Abbildung 11a) dargestellt und eignet sich aufgrund des vor dem Roboter liegenden Scanbereichs gut zur Objekterkennung und Interaktion, wie zum Beispiel das Greifen eines gesuchten Gegenstands.



a) *vor dem Roboter* b) *über dem Roboter*

Abbildung 11: Ausrichtung des Laserscanners mit Punktedichteverteilung

Abbildung 11b) zeigt die zweite Art der Ausrichtung, bei der sich der Scanner über dem Roboter befindet, mit dem toten Winkel nach unten gerichtet. Wenn die Rotation um die nach oben gerichtete Achse erfolgt, so bleibt der Roboter immer im toten Winkel und die höchste Punktintensität ist direkt über dem Aufbau.

Die zweite Ausrichtung hat im Vergleich zur ersten den entscheidenden Vorteil, dass während der Messung der gesamte Bereich vor und hinter dem Roboter abgetastet werden kann und daher weniger Messungen benötigt werden um einen kompletten Raum zu vermessen. Für die vorliegende Aufgabenstellung ist diese Ausrichtung vorzuziehen. Nur Punkte, die auf dem Boden direkt um den Roboter herum liegen werden nicht erkannt, da sie sich ebenfalls im toten Winkel befinden. Bei einer Scannerhöhe von 500 *mm* bedeutet das, dass der Boden im Abstand von 870 *mm* um den Roboter herum nicht erkannt werden kann. Durch die Bewegung des Roboters kann jedoch auch dieser Bereich erfasst werden.

Diese Arbeit verwendet die zweite Art der Ausrichtung, da sie den Anforderungen besser entspricht und einen effizienteren Aufbau zulässt.

3.3 Konstruktion

Nachdem über die Bewegung und die Ausrichtung des Laserscanners entschieden wurde, konnte die Konstruktion des Aufbaus in Angriff genommen werden. Da es sich um ein mobiles System handelt, wurden sämtliche tragende Konstruktionselemente aus leichtem Aluminium gefertigt. Trotz des geringen Gewichts des Aluminiums ist es den Ansprüchen entsprechend stabil, einfach zu bearbeiten und günstig zu beschaffen. Das Gesamtgewicht des Aufbaus beträgt mit allen Komponenten, ohne das Mikrocontrollerboard, lediglich 1,7 *kg*.

Die Konstruktion wurde mit der Studentenversion von SolidWorks 2009 erstellt, die Konstruktionszeichnungen sind in Anhang 3 zu finden. Der grundlegende Aufbau ist in Abbildung 12 als CAD-Zeichnung zu sehen.

In der Explosionsansicht in Abbildung 12a) sind die einzelnen Komponenten des Systems gut zu erkennen. Der Aufbau besteht aus einem fest auf der Roboterplattform (10) montierten Teil (4) und einem rotierenden Teil (2), der den Laserscanner (1) trägt. Der feststehende Teil besteht aus dem Schrittmotor (7), dem Schleifring (5) und der dazwischenliegenden Kupplung (6). Die Schrittmotorsteuerung (8) ist ebenfalls an dem Aufbau montiert. Der Laserscanner ist so angebracht, dass der Mittelpunkt des Scanbereichs genau auf der Verlängerung der Motorachse liegt. Das macht die Umrechnung der Messwerte in kartesische Koordinaten einfacher. Die Konstruktion des feststehenden Teils besteht aus drei vertikalen Stützen, an denen drei Ebenen befestigt sind. Die unterste hält den Motor, die mittlere den Schleifring und die oberste besitzt eine Vertiefung, in die das Nadellager (3) eingelegt werden kann. Der gesamte Aufbau ist auf eine T-förmige Grundplatte (9) aufgeschraubt, die an die Plattform des Pioneer (10) angepasst ist und sich auf sie montieren lässt. Durch Austauschen der Grundplatte kann man den Aufbau auch auf alle Pioneer Robotermodelle oder andere Roboterplattformen mit genügend Montageraum portieren.

Der in Abbildung 12d) und e) bläulich dargestellte Bereich zeigt den Scanbereich des Laserscanners, mit dem nach unten gerichteten toten Winkel von 120°.

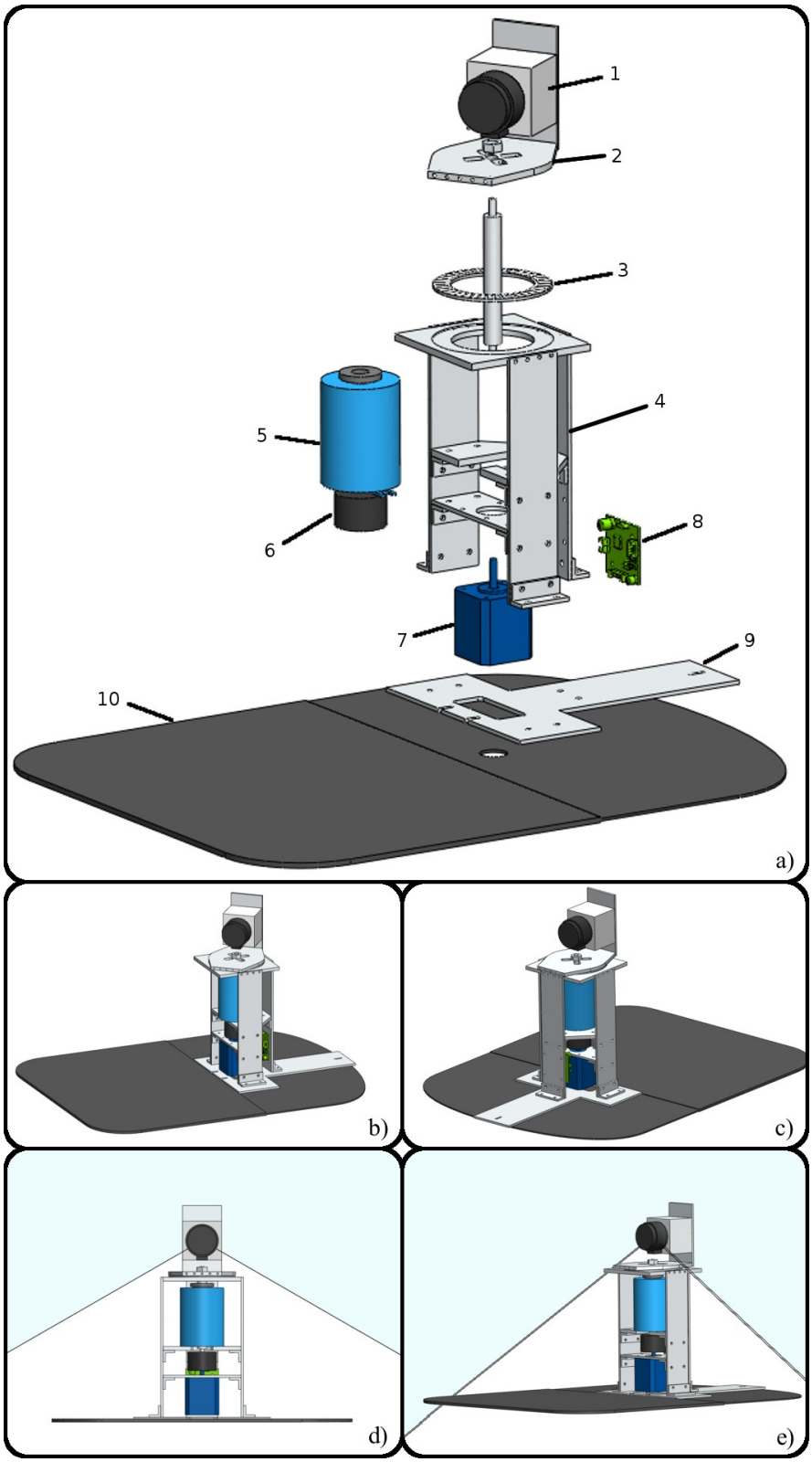


Abbildung 12: CAD-Zeichnungen der Konstruktion
a) *Explosionsansicht*
b) + c) *verschiedene Ansichten*
d) + e) *Ansichten mit dargestelltem Scanbereich*

Sowohl der rotierende, als auch der feststehende Teil der Konstruktion bietet Raum für zukünftige Erweiterungen. Die Aluminiumplatte, an der der Scanner montiert ist, ist größer ausgelegt als für diese Anwendung notwendig. Das zusätzliche Material bietet genügend Platz für die Montage einer Kamera oder weiterer Laserscanner. Auch an den vertikalen Stützen des feststehenden Teils des Aufbaus kann man ohne weiteres weitere Sensoren anbringen, wie zum Beispiel Kameras für Stereovision. Auch der großzügige Platz auf der Grundplatte eignet sich für die Montage weiterer Messinstrumente.

Das Mikrocontrollerboard wird innerhalb des Gehäuses der Roboterplattform mit einer speziell gefertigten Halteplatte befestigt und ist über die abnehmbare Gehäusewand an der Front zu erreichen. Abbildung 13 zeigt die Position des Mikrocontrollerboards in dem Pioneer.

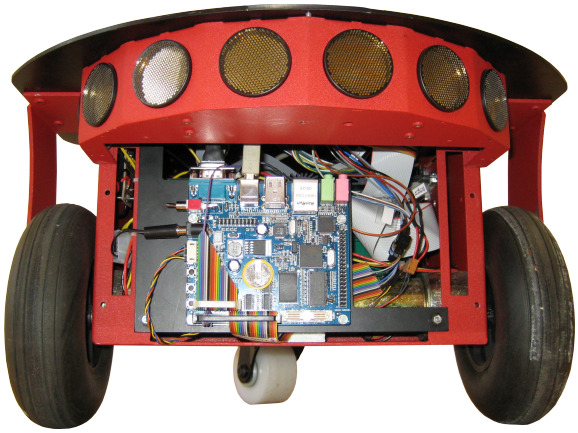


Abbildung 13: Das Mikrocontrollerboard in die Front des Pioneer eingesetzt

Der Drehmittelpunkt des Pioneer liegt ebenfalls auf der Motorachse, was wiederum die Berechnungen während der Bewegung des Pioneer vereinfacht. Die Leitungen des Schleifrings und des Schrittmotors verlaufen durch eine Öffnung in der Plattform in das Innere des Roboters, wo sie an das Mikrocontrollerboard angeschlossen sind. Die 24 Leitungen auf der oberen Seite des Schleifrings sind in vier Bündel zu je sechs Leitungen zusammengefasst und mit Flachbandsteckern abgeschlossen. Sie werden durch vier Öffnungen in der Rotationsplattform geführt und in eine dort befestigte Leiste eingesteckt.

Um die Startposition des Laserscanners genau definieren zu können wurde eine Aluminiumkonstruktion mit einer speziellen Form gefertigt, mit der man die Rotationsplattform in einer Initialposition eindeutig fixieren kann, siehe Abbildung 14. Diese Position dient den Berechnungen der kartesischen Koordinaten als Ausgangslage. Man kann darüber auch die Wiederholgenauigkeit des Systems testen und feststellen, ob und wie genau die Rotationsplattform nach einer Umdrehung wieder ihren Ausgangspunkt erreicht.

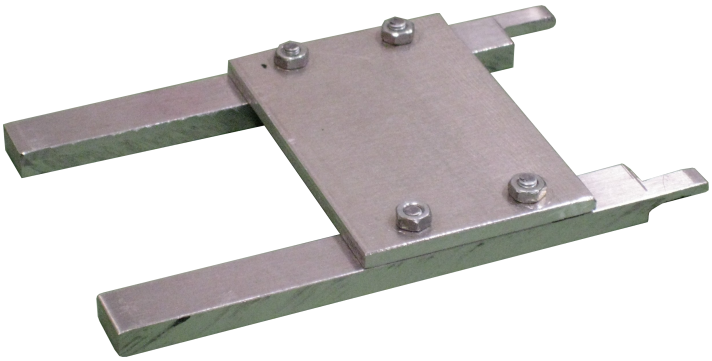


Abbildung 14: Konstruktion für die Fixierung in der Startposition

Abbildung 15 zeigt ein Foto des fertigen Aufbaus mit dem Hokuyo-Laserscanner und dem Pioneer mit geschlossener Gehäusewand, die das Mikrocontrollerboard verdeckt.

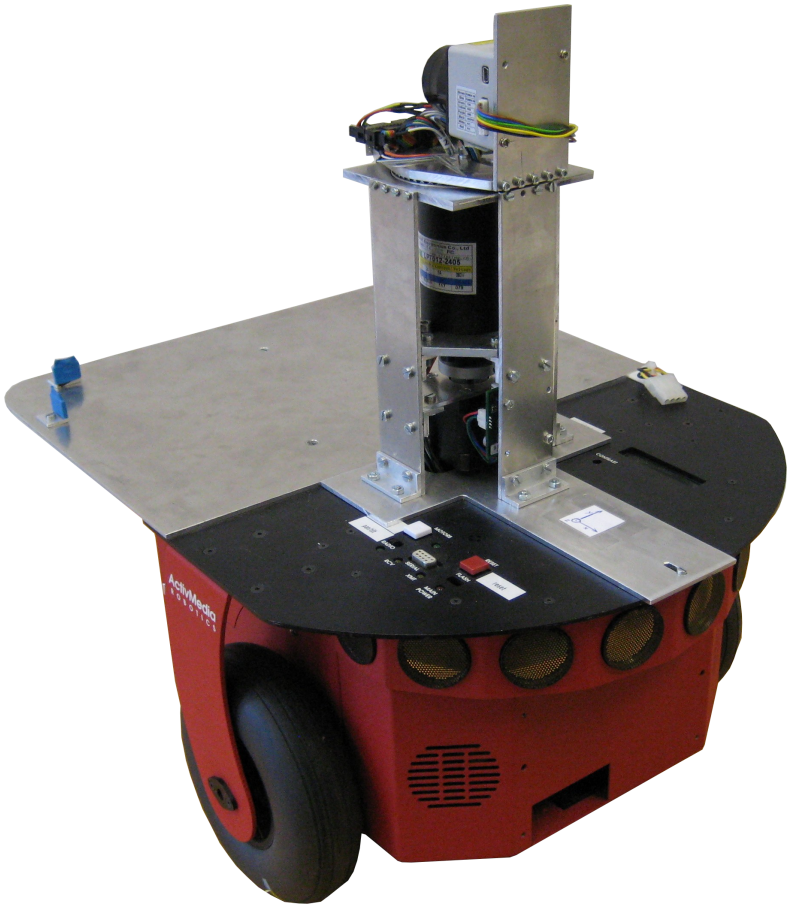


Abbildung 15: Foto des Pioneer mit dem fertigen Aufbau

Die Konstruktionselemente, die für die Erweiterung des Systems auf den IBEO-Laserscanner entworfen wurden, sind in Abschnitt IV genauer beschrieben.

4 Zusammenfassung Abschnitt I

In diesem Kapitel wurden die einzelnen Komponenten und Konstruktionen beschrieben und die Überlegungen ausgeführt, die zu dem resultierenden Aufbau geführt haben. Es wurde erläutert, dass die Anforderungen und die gewünschten Erweiterungen am besten erfüllt werden, wenn die Bewegung des Laserscanners mit Hilfe eines Schrittmotors und eines Schleifrings ohne Richtungsumkehr durchgeführt werden kann. Die Ausrichtung des Laserscanners wurde so gewählt, dass die Rotation um die vertikale Achse stattfindet und der Roboter sich während der gesamten Messung im toten Winkel des Messbereichs befindet.

Die Konstruktion wurde aus leichtem Aluminium gefertigt und ist auf andere Roboterplattformen portierbar.

II Software

1 Einleitung Abschnitt II

Die für das Projekt entworfene Software gliedert sich in zwei Teile.

Der erste Teil besteht aus der Software, die auf dem Mikrocontrollerboard läuft. Sie hat die Aufgabe die Daten vom Laserscanner zu empfangen, diese vorzuverarbeiten und sie an den PC zu senden. Außerdem soll sie den Motor der Rotationsplattform ansprechen und den Laserscanner um einen bestimmten Winkel drehen. Um den Zeitpunkt der Drehung zu ermitteln wird das Synchronisationssignal des Laserscanners genutzt, siehe Kapitel I.2.2. Dieser Teil ist in Kapitel II.2 unter dem Namen Embedded Software detailliert beschrieben.

Der zweite Teil der Software besteht aus der Hauptverarbeitung der Messdaten. Diese läuft auf dem PC ab und wurde von Gregor Michalicek entwickelt. Da sie nicht der Hauptteil dieser Arbeit ist, jedoch zu dessen Anschauung nützlich ist, sei sie in Kapitel II.3 kurz mit den dazugehörigen SLAM-Algorithmen beschrieben.

Der Abschnitt schließt in Kapitel 4 mit einer kurzen Zusammenfassung der vorangegangenen Kapitel.

2 Embedded Software

2.1 Überblick

Da das verwendete Mikrocontrollerboard SBC6000X mit einem Linux (Kernel 2.6.24) ausgeliefert wird, wurde sämtliche Software für Linux geschrieben. Zum Kompilieren wurde der mitgelieferte C-Cross-Compiler benutzt. Kapitel II.2.2 beschreibt die Veränderungen, die am Kernel vorgenommen werden mussten, damit er sämtliche gewünschte Funktionen unterstützt.

Die entworfene Software ist in den Kapiteln II.2.3 bis II.2.6 beschrieben und im Anhang als Quellcode beigefügt. Sie wurde auf einem Linux-System (Ubuntu 9.10) mit einem Texteditor geschrieben.

Nachdem das System funktionstüchtig war, wurden die Reaktionszeit gemessen und Messdaten aufgenommen.

2.2 Vorbereitung des Linux-Kernels

Um den mit dem SBC6000X mitgelieferten Linux-Kernel für die vorgesehenen Aufgaben benutzen zu können mussten zwei wichtige Änderungen vorgenommen werden. Das USB-Device musste aktiviert und die GPIOs als Interrupt konfiguriert werden.

Um die GPIOs als Interrupt konfigurieren zu können, mussten Veränderungen in der Datei *arch/arm/mach-at91/board-sbc9261.c* vorgenommen werden, die im Quellcode des mitgelieferten 2.6.24 Kernel zu finden ist. Diese bewirken, dass der verwendete GPIO-Pin als Knopf angesehen wird, der jedesmal, wenn sich die angelegte Spannung ändert, einen Interrupt auslöst. Die Veränderungen in der Datei sind durch Kommentare erkenntlich gemacht und die Datei ist zusammen mit dem Quellcode und einem Abbild des kompilierten Kernels auf der beigefügten CD zu finden.

Um das USB-Device zu aktivieren musste während der Konfiguration des Kernels folgende Einstellung vorgenommen werden: *Device Drivers → USB support → USB Gadget Support → USB Gadget Drivers (Serial Gadget (with CDC ACM support))*. Mit der Aktivierung dieses Moduls ist es möglich, den USB-Device-Anschluss wie eine serielle Schnittstelle zu benutzen. Bevor man jedoch diese Einstellungen vornimmt, muss das Konfigurationsskript */arch/arm/configs/sbc6000x_defconfig* geladen werden, das auch auf der beigefügten CD zu finden ist.

Beim Systemstart des embedded Linux muss der Befehl '*mknod /dev/ttyGS0 c 127 0*' ausgeführt werden. Dieser erstellt die Schnittstelle */dev/ttyGS0*, die wie eine serielle Schnittstelle angesprochen werden kann und am USB-Device-Anschluss liegt. Die auf dem Board befindliche Datei */etc/profile* enthält die Liste der Befehle, die beim Systemstart automatisch ausgeführt werden. Fügt man den obigen Befehl in die Datei ein, so wird die Schnittstelle automatisch erstellt und steht zur Verfügung. Die endgültige Version dieser Datei befindet sich ebenfalls auf der CD.

Auf der Host-Seite muss, wenn es sich um ein Linux-System handelt, mit dem Befehl '*sudo modprobe usbserial vendor=0x0525 product=0xA4A6*' der USB-seriell-Konverter des Mikrocontrollerboards explizit mit dem Modul usbserial eingebunden werden. Erst dann wird automatisch beim Einstecken des USB-Kabels eine Schnittstelle */dev/ttyUSBX* erstellt, über die man mit dem Board kommunizieren kann.

2.3 Empfang der Messdaten

Der Laserscanner wird an die serielle Schnittstelle COM2 (*/dev/ttyS2*) des Mikrocontrollerboards angeschlossen. Die Übertragungsgeschwindigkeit wurde, wie in Kapitel I.2.2 erwähnt, auf 500 *kbps* eingestellt. Nachdem beim Programmstart zunächst der Scanner in den SCIP2.0-Modus versetzt wird, läuft der Empfang der Daten in einer Endlosschleife ab, die durch das Senden des Buchstaben 'Q' über die USB-Schnittstelle unterbrochen wird. Die Daten werden ununterbrochen byteweise ausgelesen und nur während des Scanvorgangs gespeichert, der ebenfalls über USB-Schnittstelle mit dem Senden des Buchstaben 's' gestartet werden kann. Die gespeicherten Daten werden nicht verarbeitet und liegen im Format des Laserscanners vor [1]. Neben dieser Schleife für den Empfang der Daten gibt es noch zwei weitere Schleifen in diesem Programm, die jeweils in einem eigenen Thread liegen. Der eine ist für den Empfang der Steuersignale über die USB-Schnittstelle, der andere für das Reagieren auf das Synchronisationssignal und das Steuern der Plattform zuständig.

2.4 Steuerung der Rotationsplattform

An die serielle Schnittstelle COM1 (*/dev/ttyS1*) des Mikrocontrollerboards ist die Schrittmotorsteuerung angeschlossen. Das TMCL-Protokoll [4] ermöglicht es, den Motor direkt anzusteuern, indem man über die serielle Schnittstelle bestimmte Befehle verschickt. Neben den Befehlen für das Ändern der Motorgeschwindigkeit und -beschleunigung wird lediglich der Befehl gebraucht, der den Motor um eine bestimmte Anzahl von Mikroschritten rotieren lässt. Das Programm ermöglicht die Rotation um 16, 32, 64 und 128 Mikroschritte, die den Gradzahlen 0,45°, 0,9°, 1,8° und 3,6° entsprechen. Die Geschwindigkeit und Beschleunigung des Motors ist für die verschiedenen Schrittgrößen unterschiedlich, um eine flüssige Bewegung ohne Schwingungen zu erhalten. Die Werte werden mit Übergabewerten, Zahlen zwischen 0 und 2047, festgelegt und lassen sich mit Hilfe des Programms TMCL-IDE in die Einheit Umdrehungen pro Sekunde (*RPS*) und Umdrehungen pro Sekunde pro Sekunde (*RPS/s*) umrechnen. Tabelle 1 stellt die Werte dar, die in dem Programm gewählt wurden. Die niedrige Geschwindigkeit und Beschleunigung wurde für die größte Schrittweite von 128 gewählt, da sonst das System in eine zu starke Schwingung versetzt

wird, die dem Aufbau schaden kann. Die sich aus den niedrigeren Werten ergebende Bewegung ist flüssiger, jedoch kann die Endposition nicht erreicht werden bevor der nächste Scan beginnt, siehe Kapitel III.2. Daher sind die Messwerte bei der niedrigsten Auflösung bei weitem nicht so genau wie die der anderen Auflösungen.

Schrittgröße	Geschwindigkeit		Beschleunigung	
	Übergabewert	*RPS*	Übergabewert	*RPS/s*
16	2047	4,880	2047	74,469
32	2047	4,880	2047	74,469
64	2047	4,880	2047	74,469
128	100	0,238	100	3,637

Tabelle 1: Rotationsgeschwindigkeiten und -beschleunigungen

Das Steuern der Rotationsplattform wird in der Software in einem eigenen Thread bearbeitet. Innerhalb dieses Threads läuft eine Endlosschleife, die auf Interrupts reagiert, die ausgelöst werden, sobald das Synchronisationssignal des Laserscanners einsetzt. Sobald ein 's' über die USB-

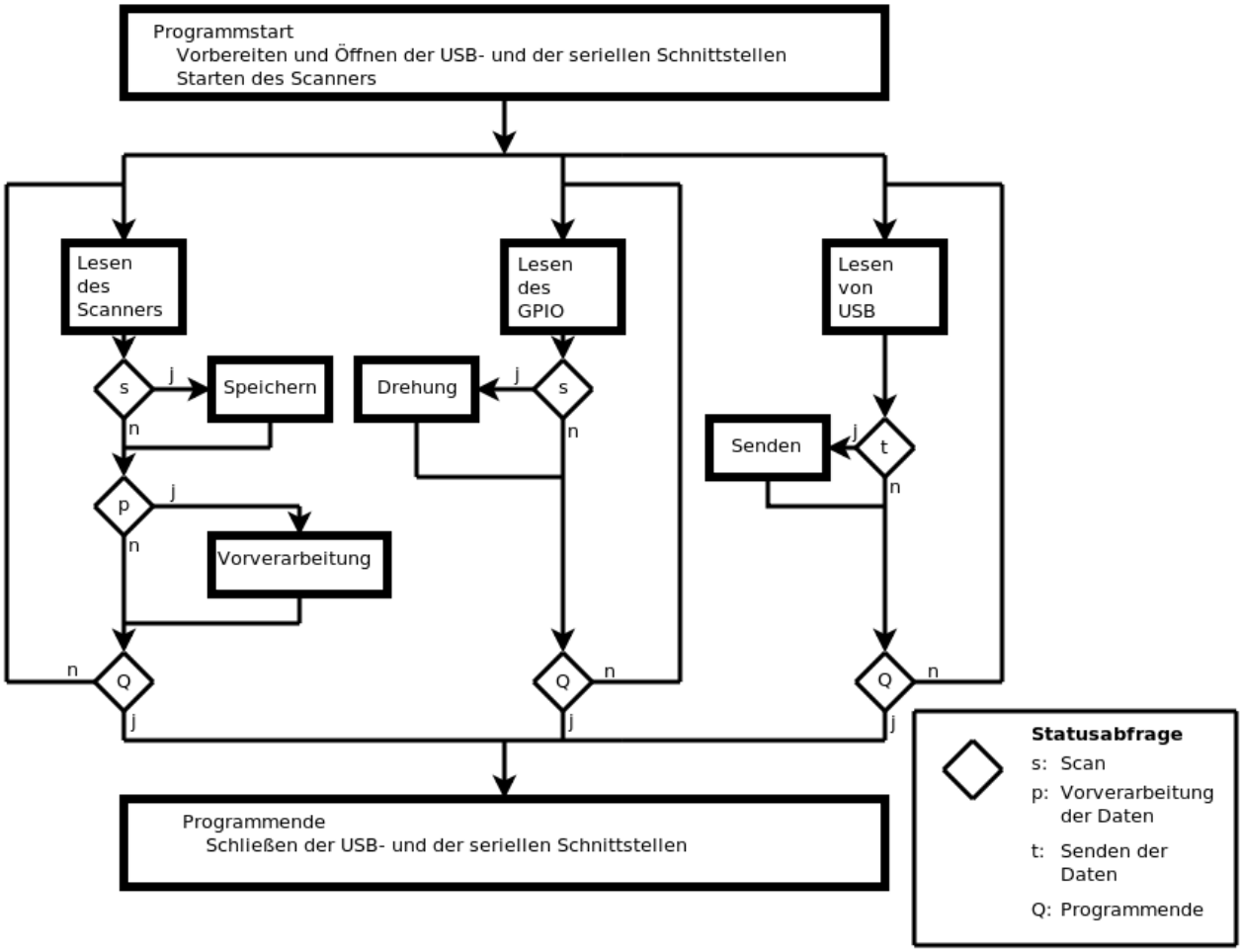


Abbildung 16: Flussdiagramm des Programms

Schnittstelle empfangen wurde, wird mit jedem Interrupt der Motor um die spezifizierte Schrittgröße gedreht. Dieser Vorgang wird wiederholt, bis die Plattform eine halbe Umdrehung ausgeführt hat, was nach 400, 200, 100 oder 50 Schritten der Fall ist, abhängig von der Schrittgröße.

Sobald die halbe Umdrehung vollendet ist, und somit rundherum alle Punkte einmal vermessen wurden, startet automatisch die Vorverarbeitung der Messdaten.

Der Verlauf des Programms ist in Abbildung 16 in einem Flussdiagramm dargestellt. Man erkennt die drei Threads zum Lesen der zwei seriellen und der USB-Schnittstelle, die ausgeführt werden, bis über die USB-Schnittstelle der Buchstabe 'Q' empfangen und damit das Programm beendet wird.

2.5 Vorverarbeitung der Messdaten

Die Vorverarbeitung der Messdaten besteht aus zwei Teilen, dem Erkennen und Dekodieren der kodierten Messdaten und dem Umrechnen der dekodierten Daten in kartesische Koordinaten.

Zunächst muss man aus der Menge der Daten, die vom Laserscanner empfangen wurden, die Messwerte herausfinden. Dabei macht man sich zunutze, dass jede Messreihe mit der Wiederholung des Scanbefehls beginnt. Durchsucht man die empfangenen Rohdaten nach diesem Befehl, so kann man die Messdaten von den Prüfsummen, die nach jeweils 64 Byte gesendet werden unterscheiden.

Es muss jedoch überprüft werden, ob die Daten ohne Übertragungsfehler gespeichert werden konnten. Dazu kann man die Stellen der Rohdaten überprüfen, die bei fehlerloser Übertragung immer das gleiche Zeichen, nämlich ein 'Linefeed' (0x0A) enthalten. Stimmt das Zeichen auf einer der Positionen nicht damit überein, so weiß man, dass die Messung Übertragungsfehler enthält. Messungen mit Übertragungsfehlern werden nicht dekodiert, sondern auf 0 gesetzt und somit für weitere Berechnungen ignoriert.

Beim Dekodieren der korrekt empfangenen Messdaten muss zunächst unterschieden werden, ob eine Zwei- oder Dreibyte-Verschlüsselung vorliegt. Bei der Zweibyte-Verschlüsselung werden jeweils zwei Character zu einer Zahl zwischen 0 und 4095 umgerechnet, bei der Dreibyte-Verschlüsselung jeweils drei Character zu einer Zahl zwischen 0 und 5600. Die Kodierung ist genau in der Beschreibung des SCIP2.0 Protokolls [1] dargestellt. Die Zahl gibt die Entfernung des Messpunktes in Millimetern an. Durch das Programm werden 682 Messpunkte pro Ebene ausgelesen, was dem maximalen Öffnungswinkel des Laserscanners entspricht. Die Punkte liegen etwa 0,35° (360°/1024) auseinander und der erste Punkt liegt bei circa 60,5° (172 x 360°/1024), der letzte bei -60,5°, wenn man den Winkel von der nach unten gerichteten Achse angibt.

Aus den dekodierten Messdaten lassen sich dann die kartesischen Koordinaten der Messpunkte errechnen. Mittelpunkt des, wie in der Robotik üblichen, rechtshändigen kartesischen Koordinatensystems (x, y, z) ist in diesem Fall der Mittelpunkt des Scanners, die x-Achse ist nach vorne, die z-Achse nach oben gerichtet. Mittelpunkt des bewegten Koordinatensystems (x', y', z') ist ebenfalls der Mittelpunkt des Scanners, die z-Achse zeigt wieder nach oben, doch die x-Achse zeigt während der gesamten Bewegung zum Fuß des Scanners. Die beiden Koordinatensysteme sind im Bezug zueinander und zur Konstruktion in Abbildung 17 dargestellt. Der Verdrehwinkel der Rotationsplattform wird in dieser Arbeit mit β bezeichnet.

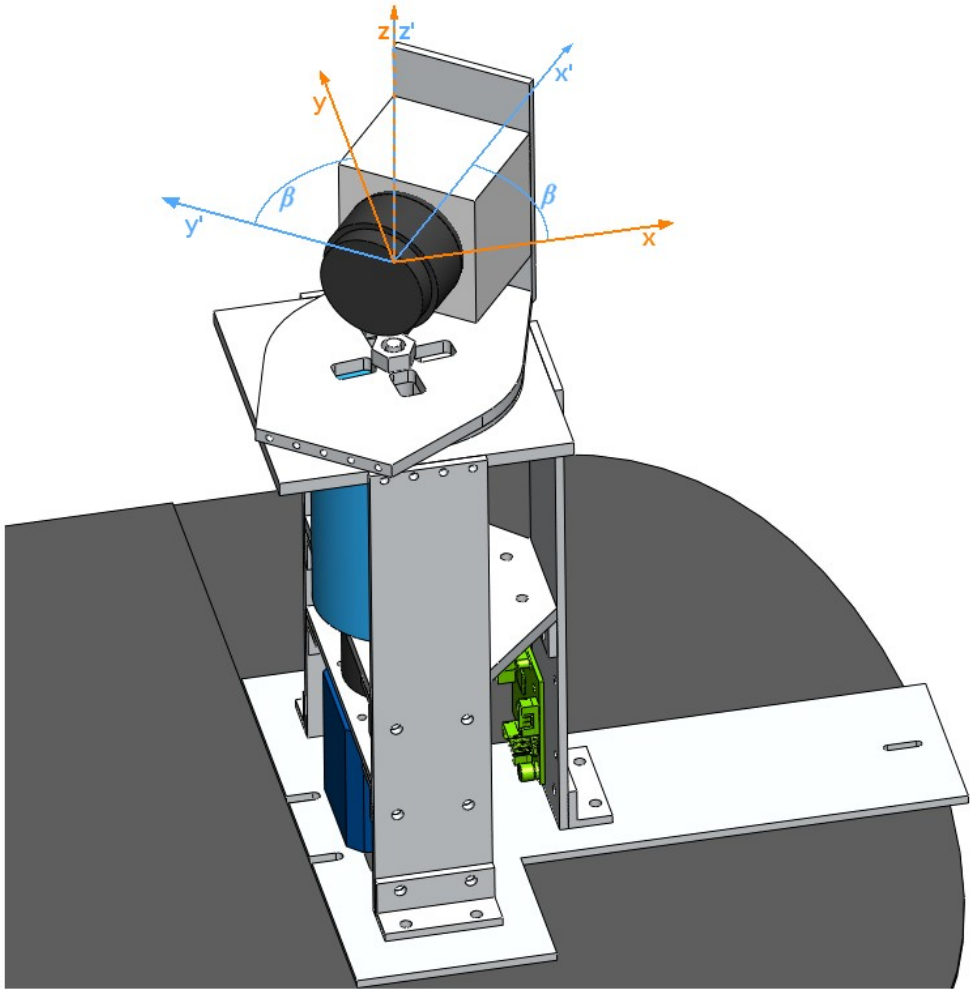


Abbildung 17: Position des bewegten (blau) und des feststehenden (orange) Koordinatensystems

Um die Formel der Koordinatentransformation zu bestimmen wurde zunächst die Koordinate des jeweiligen Messpunktes in dem mit der Scanebene mitrotierenden Koordinatensystem (x', y', z') bestimmt.

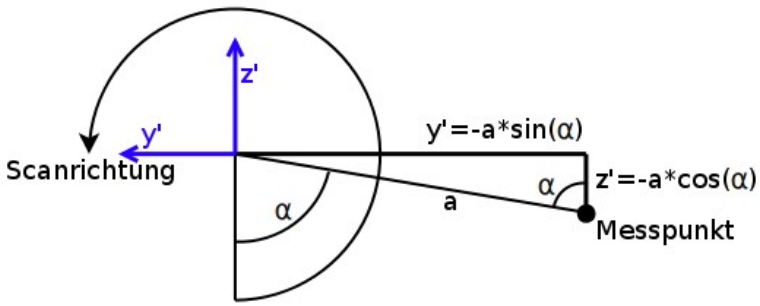


Abbildung 18: Berechnung der Koordinaten im bewegten Koordinatensystem (x', y', z')

Betrachtet man die Scanebene wie in Abbildung 18 dargestellt mit dem Scanwinkel α und dem Entfernungsmesswert *a*, so ergibt sich für die x'-, y'- und z'-Koordinaten folgender Vektor:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} 0 \\ -a\sin(\alpha) \\ -a\cos(\alpha) \end{pmatrix} \tag{6}$$

Um die Rotation der Plattform um die z-Achse in die Koordinatentransformation mit einzubeziehen muss der soeben berechnete Vektor mit der Drehmatrix aus Formel 6 multipliziert werden. Dabei wird eine Drehung um den Winkel β durchgeführt [13]:

$$\begin{pmatrix} \cos(\beta) & -\sin(\beta) & 0 \\ \sin(\beta) & \cos(\beta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{7}$$

Es ergibt sich für die Berechnung der kartesischen Koordinaten folgende Formel:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \sin(\alpha)\sin(\beta)a \\ -\sin(\alpha)\cos(\beta)a \\ -\cos(\alpha)a \end{pmatrix} \tag{8}$$

Mit *i* als der Nummer des aktuellen Scans, *s* der des aktuellen Motorschritts und *m,* der Gesamtanzahl der Schritte, lassen sich α und β berechnen:

$$\begin{aligned} \alpha &= 60,5\,° + i * 0,35\,° \\ \beta &= s * \frac{180\,°}{m} \end{aligned} \tag{9}$$

Die Berechnung der kartesischen Koordinaten wird jedoch nur dann ausgeführt, wenn der Entfernungsmesswert *a* größer als 20 *mm* und kleiner als die maximale Reichweite ist, die bei der Zweibyte-Verschlüsselung 4095 *mm* und bei der Dreibyte-Verschlüsselung 5600 *mm* ist. Liegt der Wert außerhalb dieser Grenzen, so handelt es sich um eine Fehlmessung und der Wert wird ignoriert, siehe Kapitel I.2.2.

Allerdings muss noch geprüft werden, in welche Richtung der Scanner bei Beginn der Rotation ausgerichtet ist. Da es sich immer um halbe Umdrehungen handelt, die der Scanner bei einem Scan vollführt, muss bei jeder zweiten Messung ein Vorzeichenwechsel der x- und y-Koordinate erfolgen.

Die erste Version der Software hat sämtliche Daten, sowohl die Rohdaten, als auch die verarbeiteten Daten, in Textdateien zwischengespeichert, was aufgrund des relativ langsamen Dateizugriffs zu einer hohen Verarbeitungszeit geführt hat. Eine Optimierung des Programms, bei der unter anderem die Daten statt in Textdateien in Variablen gespeichert wurden brachte einen erheblichen Zeitgewinn, wie man Tabelle 2 entnehmen kann.

Schrittgröße	Berechnung der Polar- und kartesischen Koordinaten	
	vor der Optimierung	nach der Optimierung
128	17 *s*	4 *s*
64	33 *s*	9 *s*
32	1 *Min* 5 *s*	18 *s*
16	2 *Min* 13 *s*	38 *s*

Tabelle 2: Verarbeitungszeiten vor und nach der Optimierung

2.6 Versand der Daten

Man kann mit dem Programm drei verschiedene Datensätze über die USB-Schnittstelle an einen PC versenden: die unverarbeiteten Rohdaten, wie sie der Laserscanner während der Messung verschickt, die dekodierten Daten, die nur die Polarkoordinaten in Paketen zu je 682 Messwerten enthalten und die kartesischen Koordinaten der erfolgreich vermessenen Punkte.

Um die Übertragungszeit zu verringern, wurden die Polarkoordinaten und die kartesischen Koordinaten wiederum kodiert und so deren benötigter Speicherplatz verringert. Im Gegensatz zur Kodierung die Hokuyo mit dem Laserscanner vornimmt, lassen sich mit dieser Kodierung jedoch sämtliche Messwerte zwischen 0 und 6400 in zwei Byte darstellen, einem hohen und einem niedrigen Byte. Deren Berechnung kann auf folgende Weise durchgeführt werden:

hohes Byte: Messwert / 80 + 0x28
niedriges Byte: Messwert % 80 + 0x28

Dabei stehen die Operatoren '/' für die Division ohne Rest und '%' für den Modulo, also den Rest bei entsprechender Division und die 0x28 für die hexadezimale Darstellung der Zahl 40. Die sich ergebende (hexadezimale) Zahl kann man mit der Verwendung einer ASCII-Tabelle [21] als Character darstellen und versenden. Die ersten Zeichen in der ASCII-Tabelle entsprechen Steuerzeichen, die beim Übertragen Schwierigkeiten machen können. Durch die Addition der 0x28 ist gewährleistet, dass kein Steuerzeichen übertragen werden muss.

Der Vorgang sei hier nochmal an dem Beispiel des Messwertes 5321 verdeutlicht.

hohes Byte: 5321 / 80 + 0x28 = 66 + 0x28 = 0x42 + 0x28 = 0x6A

niedriges Byte: 5321 % 80 + 0x28 = 41 + 0x28 = 0x29 + 0x28 = 0x51

In der ASCII-Tabelle entspricht die 0x6A dem 'j' und die 0x51 dem 'Q', also wird die Zahl 5321 als 'jQ' über die USB-Schnittstelle übertragen.

Da bei der Übertragung der kartesischen Koordinaten auch hin und wieder negative Zahlen übertragen werden müssen, wird vor der Kodierung das Vorzeichen überprüft. Handelt es sich tatsächlich um eine negative Zahl, so findet bei dem hohen Byte statt einer Addition mit 0x28 eine Addition mit 0xA8 statt, was eine eindeutige Identifizierung des Vorzeichens beim Dekodieren zulässt.

Schrittgröße	Versand der Rohdaten		Versand der Polarkoordinaten		Versand der kartesischen Koordinaten	
	ohne Kodierung	mit Kodierung	ohne Kodierung	mit Kodierung	ohne Kodierung	mit Kodierung
128	4 *s*	2 *s*	*7 s*	3 *s*	*24 s*	6 *s*
64	7 *s*	4 *s*	15 *s*	7 *s*	48 *s*	13 *s*
32	13 *s*	9 *s*	30 *s*	14 *s*	1 *Min* 35 *s*	27 *s*
16	25 *s*	18 *s*	1 *Min*	27 *s*	3 *Min*	53 *s*

Tabelle 3: Versandzeiten mit und ohne Kodierung

Die Zeitersparnis, die durch das Kodieren der Daten gewonnen wurde kann man Tabelle 3 entnehmen. Man sieht, dass die Übertragung teilweise bis zu dreimal schneller vonstatten geht, wenn die Daten vor dem Versenden kodiert wurden. Ein weiterer Vorteil der Kodierung ist auch der kleinere Speicherplatz, den die versendeten Daten benötigen.

3 Verarbeitung der Daten (SLAM)

3.1 Überblick

Lokalisierung und Navigation ist für einen mobilen autonomen Roboter eine Grundvoraussetzung, er muss schließlich wissen, wo er ist und wie er an einen anderen Ort hinkommt. Hat er einmal seine Position mit Hilfe eines Rundumscans von der Umgebung bestimmt, so kann er jedoch nicht ohne weiteres bis zu seinem Zielpunkt navigieren, ohne auf dem Weg weitere Positionsbestimmungen durchzuführen. Aufgrund von vielfältigen Umständen kommt es bei der Bewegung von Robotern immer zu Ungenauigkeiten. Das können leicht unterschiedliche Raddurchmesser oder Achsabstände, durchdrehende Reifen oder ungenaue Motorgeschwindigkeiten und vieles mehr sein. Auch die durch das System aufgenommene dreidimensionale Karte beinhaltet Ungenauigkeiten, zum Beispiel durch ungenaue Ausrichtung und die Messtoleranz des Scanners, die berücksichtigt werden sollten.

Um diese Ungenauigkeiten in der Kartenerstellung und Positionsbestimmung zu kompensieren wurde eine Vielzahl von Verfahren entwickelt, die unter dem Oberbegriff *Simultane Lokalisierung und Kartenerstellung* (SLAM[5]) zusammengefasst sind. Die Auswirkung, die die Anwendung eines SLAM-Algorithmus auf Messdaten haben kann, ist Abbildung 19 sehr gut zu entnehmen. Man erkennt in Abbildung 19 a), dass die Wände eine Krümmung besitzen, die darauf zurückgeht, dass der verwendete Roboter in einer mehr oder weniger leichten Kurve gefahren ist, obwohl er hätte geradeaus fahren sollen. Wegen der Inkrementalgeber an den Achsen, die eine gerade Fahrt bestätigt haben, wurde eine falsche Position berechnet und die Lasermessdaten mit einem falschen Bezugspunkt verknüpft. Die mit einem SLAM-Algorithmus bearbeiteten Messdaten in Abbildung 19 b) zeigen die Krümmung nicht mehr an und geben somit eine genauere Karte wieder.

5 Simultaneous Localisation and Mapping

a) *vor Anwendung eines SLAM-Algorithmus* b) *nach Anwendung eines SLAM-Algorithmus*

Abbildung 19: Darstellung von 2D-Messdaten [20]

Die Verarbeitung der Messdaten mit Hilfe von SLAM-Algorithmen ist nicht direkt Gegenstand dieser Arbeit. Da jedoch ohne Zweifel die Verwendung des Systems für die Kartenerstellung bevorsteht, folgt an dieser Stelle eine sehr kurze Einführung in das Thema. Für tiefergehende Informationen sei auf die Literaturen [17] und [20] verwiesen.

Die am häufigsten verwendete Lösung des SLAM-Problems zielt auf das Abschätzen der A-posteriori-Wahrscheinlichkeitsdichteverteilung über alle möglichen Landkarten θ und alle möglichen Posen des Roboters s_t zum Zeitpunkt t hin. Abhängig ist diese Verteilung von sämtlichen bis dahin vollzogenen Messungen z^t und allen bisherigen Steuersignalen u^t, die der Roboter erhalten hat. Diese spezielle Verteilung wird als SLAM-Posterior[6] bezeichnet.

$$p(s_t, \theta | z^t, u^t) \tag{10}$$

Eine Landkarte θ wird meist als eine Menge von charakteristischen Punkten, sogenannten Landmarken, angegeben. Mit Pose ist die eindeutige Position des Roboters gemeint, die im zweidimensionalen Raum aus zwei Koordinaten und einem Winkel für die Ausrichtung besteht, im dreidimensionalen aus drei Koordinaten und drei Winkeln.

In Formel 10 wird angenommen, dass der Zusammenhang zwischen den Messungen z^t und den Landmarken in θ unbekannt ist. Wenn man jedoch davon ausgeht, dass die einzelnen Landmarken eindeutig voneinander unterschieden werden können, kann man das SLAM-Posterior umschreiben zu:

$$p(s_t, \theta | z^t, u^t, n^t) \tag{11}$$

Hierbei steht n^t für die Nummer der Landmarke, die zum Zeitpunkt t erkannt wurde.

Mit Hilfe eines sogenannten Bayes-Filters kann man das SLAM-Posterior für den Zeitpunkt t berechnen, wenn es für den Zeitpunkt $t-1$ bekannt ist:

$$p(s_t, \theta | z^t, u^t, n^t) = \eta\, p(z_t | s_t, \theta, n_t) \int p(s_t | s_{t-1}, u^t)\, p(s_{t-1}, \theta | z^{t-1}, u^{t-1}, n^{t-1})\, ds_{t-1} \tag{12}$$

Da im Allgemeinen das Integral in Formel 12 nicht in geschlossener Form gelöst werden kann, werden verschiedene Näherungsverfahren verwendet um es zu bestimmen und damit das SLAM-Posterior zu berechnen.

6 Im SLAM-Posterior stehen tiefgestellte Indizes für aktuelle Werte, hochgestellte für alle bis zu dem Zeitpunkt vorhandenen Werte

4 Zusammenfassung Abschnitt II

Dieser Abschnitt befasste sich mit der Softwareentwicklung, die einen großen Teil dieser Arbeit ausgemacht hat.

Es wurde beschrieben, auf welche Weise der Linux-Kernel vorbereitet und konfiguriert werden musste, um die gewünschten Schnittstellen und Interrupts benutzen zu können.

Der Empfang der Daten vom Laserscanner und das Herausfiltern der Messwerte wurden kurz angerissen. Desweiteren wurde ausführlich auf die Vorverarbeitung der Messdaten, also die Umrechnung von Polar- in kartesische Koordinaten, eingegangen. Außerdem wurde das Kodierverfahren beschrieben, das verwendet wurde, um eine Zeitersparnis beim Versenden der Messwerte zu erhalten.

Zuletzt wurde ein kurzer Überblick über das SLAM-Problem gegeben.

III Messungen

1 Einleitung Abschnitt III

Um zu gewährleisten, dass die Messwerte und die berechneten Koordinaten der Wirklichkeit entsprechen, muss gezeigt werden, dass die Plattform wie gewünscht arbeitet.

Dazu gehört die Messung der Reaktionszeit der Rotationsplattform, die so kurz sein muss, dass alle Aufgaben abgearbeitet werden können bevor die nächste Messung beginnt. Diese Messungen sind in Kapitel 2 beschrieben.

Zusätzlich wurden Versuche durchgeführt, um die Wiederholgenauigkeit des Systems zu testen. Die Ergebnisse kann man in Kapitel 3 nachlesen.

In Kapitel 4 werden schließlich einige Messergebnisse dargestellt, die der Laserscanner nach der Rotation liefert. Hier sollen die Unterschiede der verschiedenen Schrittauflösungen aufgezeigt werden, um einen Eindruck der Genauigkeit zu vermitteln.

2 Reaktionszeit der Rotationsplattform

Um die Reaktionszeit des Systems zu messen, wurde ein Oszilloskop verwendet, mit dem sich die Verzögerung zwischen dem Synchronisationssignal und dem Versand des Motorbefehls messen lässt. Das Gerät wurde dafür an den Interrupt-Pin und dem Sende-Pin (TxD) der seriellen Schnittstelle COM1 (*/dev/ttyS1*) angeschlossen.

Es wurde eine durchschnittliche Verzögerung von 0,4 *ms* beobachtet, bis der Befehl zum Bewegen des Motors gesendet wurde. In unregelmäßigen Abständen von einigen Sekunden verlängerte sich die Verzögerung kontinuierlich bis auf 4 *ms*, was auf Rechenprozesse auf dem Board zurückzuführen ist, die die Interruptverarbeitung verzögern. Das Versenden des Befehls dauert etwa 1,7 *ms*. Man kann also mit großer Wahrscheinlichkeit garantieren, dass der Bewegungsbefehl spätestens 6 *ms* nach dem Einsetzen des Synchronisationssignals bei dem Motor angelangt ist.

Bedenkt man, dass die Zeitspanne zwischen dem Einsetzen des Synchronisationssignals und dem Start des neuen Scans ungefähr 29 *ms* beträgt, so bedeutet das, dass dem Motor etwa 23 *ms* zur Verfügung stehen, um seine nächste Position zu erreichen.

Mit den in Kapitel II.2.4 aufgezeigten Geschwindigkeiten und Beschleunigungen lässt sich die Zeit berechnen, die für das Erreichen der nächsten Position benötigt wird. Tabelle 4 zeigt die Zeit, die der Motor benötigt, um einen Schritt mit der berechneten Beschleunigung und Geschwindigkeit zu verfahren.

Um die benötigte Zeit zu berechnen, wurde für die drei höchsten Auflösungen folgende Formel verwendet, mit β, dem Schrittwinkel im Bogenmaß, und a, der Beschleunigung in *RPS/s* [13]:

$$t = 2\sqrt{\frac{\beta}{a}} \tag{13}$$

Die Geschwindigkeit musste bei der Berechnung der Laufzeit nicht berücksichtigt werden, da sie nicht erreicht wurde, bevor der Motor abbremste.

Da die angegebene Geschwindigkeit bei der 3,6°-Auflösung jedoch erreicht wurde, musste erst mit

$$t_a = \frac{v}{a} \tag{14}$$

die Zeit berechnet werden, die zum Erreichen der Geschwindigkeit v benötigt wurde und dann der Winkel (im Bogenmaß), der bis dahin zurückgelegt worden war.

$$\beta_a = \frac{1}{2} a t_a^2 \tag{15}$$

Daraus ließ sich die Zeit ermitteln, während der mit konstanter Geschwindigkeit verfahren wurde:

$$t_v = \frac{\beta - 2\beta_a}{v} \tag{16}$$

Die Gesamtdauer der Bewegung ergab sich aus der Beschleunigungszeit addiert mit der soeben berechneten Dauer:

$$t = 2t_a + t_v \tag{17}$$

Schrittgröße in Grad	Geschwindigkeit (*RPS*)	Beschleunigung (*RPS/s*)	benötigte Zeit (*ms*)
0,45°	4,880	74,469	20
0,9°	4,880	74,469	29
1,8°	4,880	74,469	41
3,6°	0,238	3,637	329

Tabelle 4: Motordrehzeiten bei verschiedenen Schrittgrößen

Wie man in Tabelle 4 sieht benötigt der Motor in der höchsten Auflösung unter 23 *ms*, um die Bewegung durchzuführen. Daher kann man davon ausgehen, dass er schon seine Sollposition erreicht hat, wenn der nächste Scan beginnt. In der 0,9°-Auflösung ist der Motor mit 29 *ms* nicht schnell genug seine Position zu erreichen. Mit dem Übergabewert von 2047 ist jedoch die maximale Beschleunigung bereits erreicht. Die Folge dieser etwas zu langsamen Motorbewegung ist, dass Messwerte, die am Anfang eines Scans liegen mit der Bewegung des Motors verfälscht sind. In den Resultaten fällt diese leichte Ungenauigkeit jedoch nicht auf, weil der Motor seine Position schon fast erreicht hat, wenn der Scan beginnt. Das bedeutet, dass anstatt dem geplanten Messpunkt ein direkter Nachbarpunkt gemessen wird und deren Entfernungen sind in den meisten Fällen nur sehr leicht verschieden.

Ähnlich verhält es sich mit der 1,8°-Auflösung, bei der die Motorbewegung 41 *ms* dauert. Hier sind allerdings etwas mehr Messwerte zu Beginn jedes Scans verfälscht. Jedoch ist auch hier der Großteil der Messwerte unverfälscht und die verfälschten Punkte liegen nahe an den geplanten Messwerten, wodurch man die Verfälschung vernachlässigen kann.

Die geringste Auflösung zeigt mit 329 *ms* eine deutliche Verzögerung des Motors, sodass dieser während der Messung nicht zum Stehen kommt, da der Bewegungsbefehl bereits dreimal empfangen wurde, bevor die erste Position überhaupt erreicht ist. Dass der Motor dennoch genau 180° dreht und kein Befehl verloren geht ist der Schrittmotor-Ansteuerungseinheit zu verdanken.

Die Messergebnisse bei niedrigster Auflösung ergeben ein grobes Bild der Umgebung, jedoch sollte man nicht Einzelheiten der Messung vertrauen. Eine Kompensierung der Bewegung während des Scans, wie im Ausblick erwähnt, ist für diese Auflösung empfehlenswert.

3 Wiederholgenauigkeit der Rotationsplattform

Die Wiederholgenauigkeit gibt an, wie genau eine Sollposition angefahren werden kann und wie sich die wirkliche Position über mehrere Positionierungen verhält. In dieser Arbeit heißt das: Wie genau wird die Ausgangslage wieder erreicht, wenn die Rotationsplattform eine, zwei oder mehrere Umdrehungen vollführt hat.

Um dies festzustellen, wurden mehrere Scans hintereinander ausgeführt und bei jeder vollen Umdrehung getestet, ob die Ausgangsposition erreicht wurde oder ob der Schrittmotor eine falsche Anzahl von Schritten gedreht hat. Als Messwerkzeug dient die speziell gefertigte Konstruktion, mit der man die Rotationsplattform in der Ausgangslage fixieren kann, siehe Kapitel I.3.3. Diese Methode ist leider nicht sehr genau und deshalb sind die Ergebnisse mit Vorsicht zu genießen. Ein wie im Ausblick erwähnter Inkrementalgeber kann bei der Bestimmung der Wiederholgenauigkeit sehr nützlich sein.

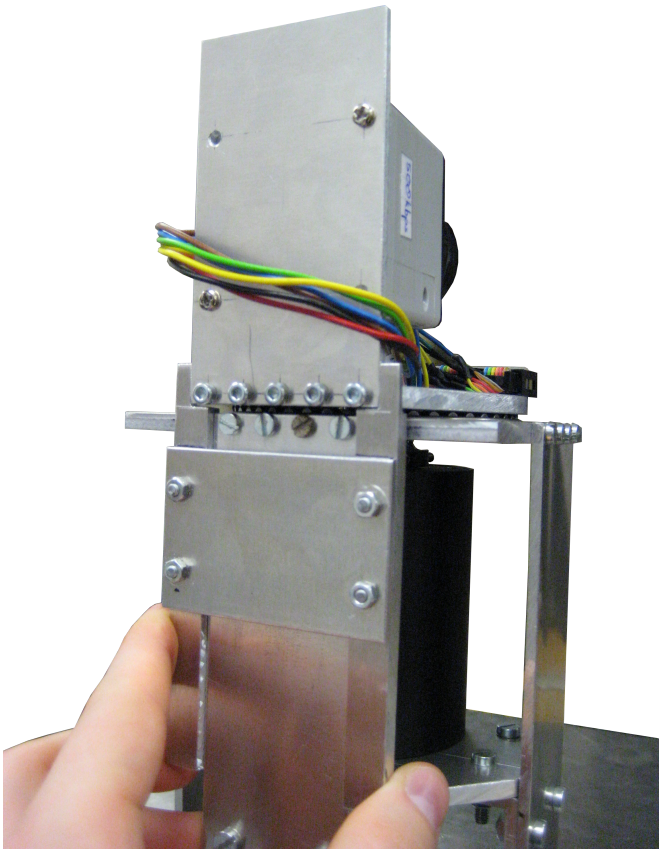


Abbildung 20: Foto der fixierten Rotationsplattform

Es hat sich gezeigt, dass, da der Motor keine Informationen über verlorene Schritte erhält, bei einem starken Widerstand die Sollposition nicht erreicht werden kann. Dies sollte bei der Verwendung von schweren Lasten auf jeden Fall berücksichtigt werden.

Bei der Verwendung des URG04-LX spielt das jedoch keine Rolle. Unbedingt zu beachten ist jedoch, dass die Plattform direkt beim Einschalten des Roboters mit der Vorrichtung fixiert wird. Geschieht das nicht und wird die Plattform erst dann ausgerichtet, wenn der Motor schon mit Strom versorgt ist, erhöht das die Wahrscheinlichkeit, dass Schritte verloren gehen. Eine weitere gute Möglichkeit die Wiederhohlgenauigkeit zu erhöhen, besteht darin die Kupplung auf einer Seite zu lockern und einen Scan zu starten. Ist dieser vorbei, kann die Kupplung wieder festgezogen werden und der Motor ist mit der Plattform gut ausgerichtet. Wird das beachtet, so fährt die Plattform mit dem URG04-LX recht zuverlässig in die Ausgangsposition zurück.

Es wurden jeweils zwanzig Scans, also zehn Umdrehungen nacheinander ausgeführt und jeweils die Position überprüft. Es hat sich gezeigt, dass die Bewegung mit der niedrigsten Auflösung, also 3,6° am fehleranfälligsten ist. Hier kam es sehr häufig zu Schrittfehlern und es wurden keine zehn Umdrehungen einwandfrei geschafft.

Bei höheren Auflösungen hat sich ergeben, dass die meisten Schrittfehler am Anfang der Messungen entstehen. Nachdem die Rotationsplattform zwei Umdrehungen ohne Schrittfehler ausgeführt hatte, kam es bei dem Test zu keinen Fehlern mehr.

4 Visualisierung der Messergebnisse

Wie in Abschnitt II erwähnt, kann man mit der entworfenen Software die Rotationsplattform in vier verschiedenen Auflösungen und zwei verschiedenen Reichweiten betreiben. Die Abbildungen 21

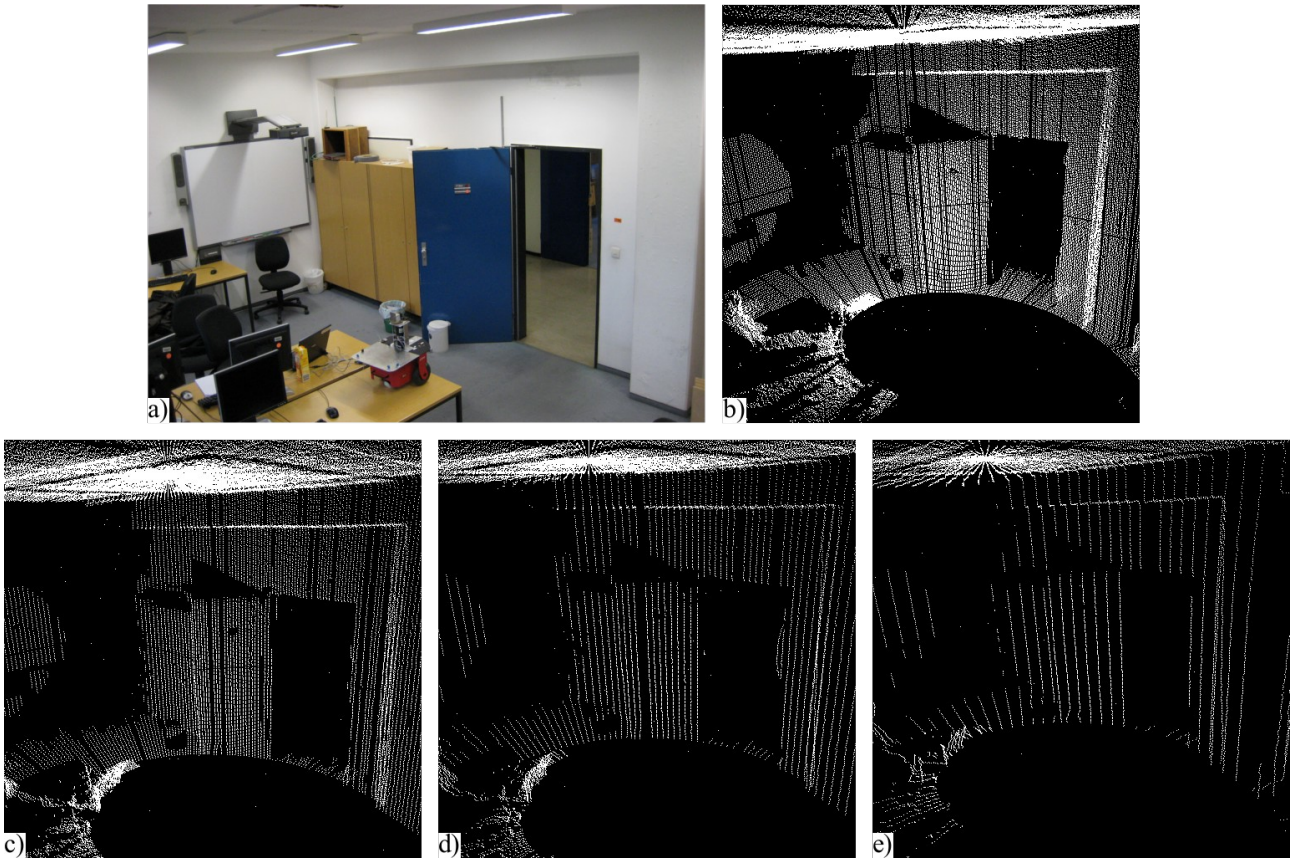


Abbildung 21: Darstellung der Messwerte mit niedriger Reichweite
a) *Foto* b) *Schrittweite 0,45°*
c) *Schrittweite 0,9°* d) *Schrittweite 1,8°* e) *Schrittweite 3,6°*

und 22 zeigen die Ergebnisse der Messreihen. Bild a) zeigt jeweils ein Foto des Arbeitsraums, die Bilder b) bis e) zeigen eine Punktewolkendarstellung des Raums aus derselben Perspektive in verschiedenen Auflösungen. Bei der Punktewolkendarstellung werden die berechneten kartesischen Koordinaten mit einem Programm in einem leeren dreidimensionalen (schwarzen) Raum als (weiße) Punkte dargestellt, um so ihre Position zueinander einschätzen zu können. Das Programm zur Darstellung der Punktewolke als dreidimensionales Bild wurde von Gregor Michalicek geschrieben und befindet sich auf der beigelegten CD.

In der Darstellung der Punktewolken erkennt man sofort den großen Unterschied zwischen den Auflösungen. Während man mit der höchsten Auflösung sogar Einzelheiten genau erkennen kann, sind bei der niedrigsten Auflösung die Details nur zu erahnen. Man kann auch leicht Fehlmessungen des Scanners erkennen, da sie ohne direkte Nachbarpunkte mitten im Raum zu schweben scheinen. Es ist auch deutlich an der in der Punktewolke nur teilweise dargestellten hinteren Wand zu erkennen, wie weit die Reichweite des Scanners ist. Mit der höheren Reichweite wurde, wie in Abbildung 22 zu sehen, die Wand komplett vermessen.

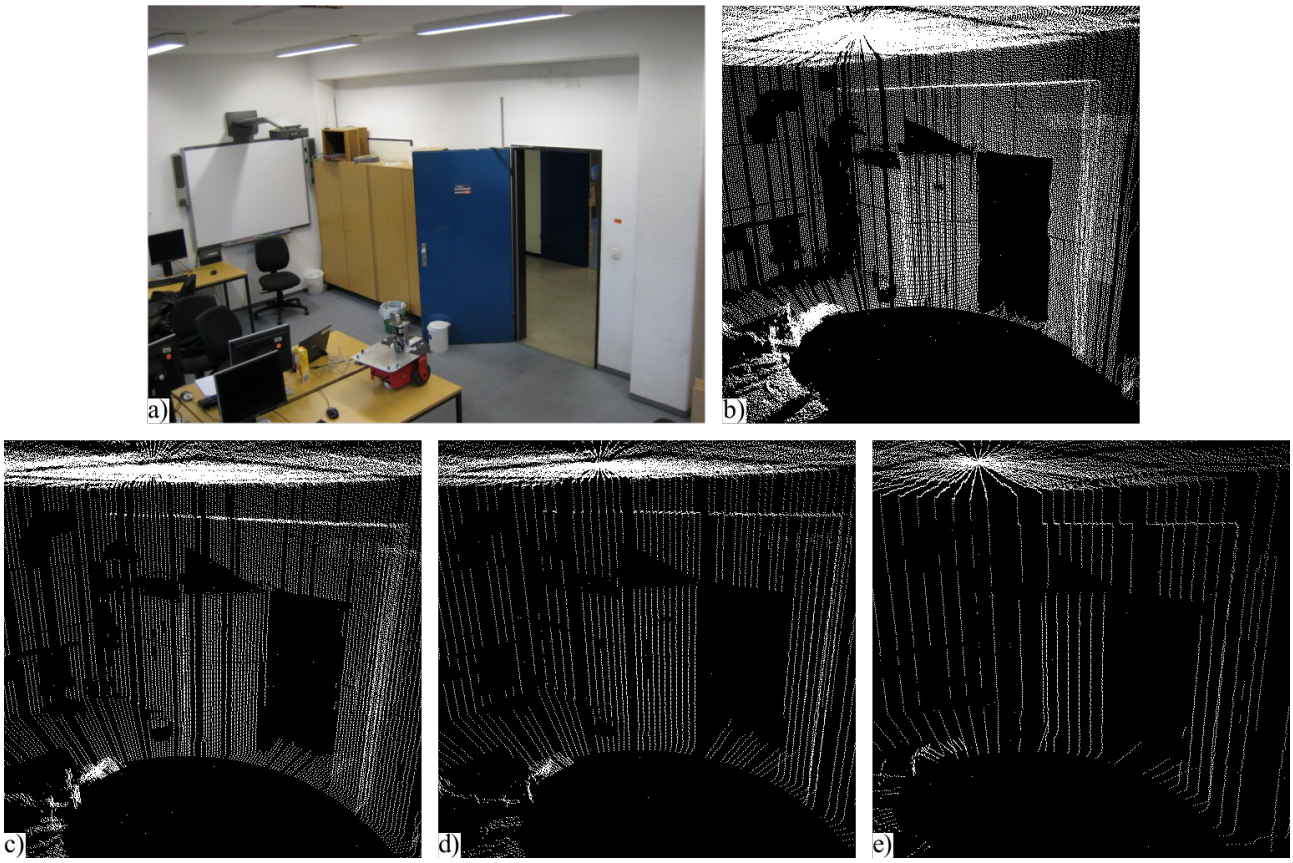


Abbildung 22: Darstellung der Messwerte mit hoher Reichweite
a) *Foto* b) *Schrittweite 0,45°*
c) *Schrittweite 0,9°* d) *Schrittweite 1,8°* e) *Schrittweite 3,6°*

Auch in Abbildung 22 sieht man den großen Unterschied zwischen den Auflösungen. Hier offenbart sich jedoch auch deutlich der Nachteil des Lasermessverfahrens bezüglich der Oberflächenbeschaffenheit der gemessenen Objekte. Der Beamer, der an der hinteren Wand über der Tafel hängt, scheint sämtliche Laserstrahlen zu verschlucken, da an seiner Stelle keine

Messwerte vorliegen. Genauso verhält es sich auch mit dem Bürostuhl, der unter der Tafel steht.

Um aus den Punktewolken Oberflächen zu rekonstruieren, wurden die Messwerte mit Hilfe des sogenannten Ball-Pivoting-Algorithmus [19] bearbeitet. Oberflächenrekonstruktion ist zum Beispiel dann wichtig, wenn man bei der Erstellung von virtuellen Realitäten Texturen auf die Oberflächen legen will. Auch für die Pfadfindung ist es einfacher mit Körpern und Oberflächen zu arbeiten als mit Punktewolken. Dieser Algorithmus wurde ebenfalls von Gregor Michalichek realisiert und die Ergebnisse sind in Abbildung 23 zu finden.

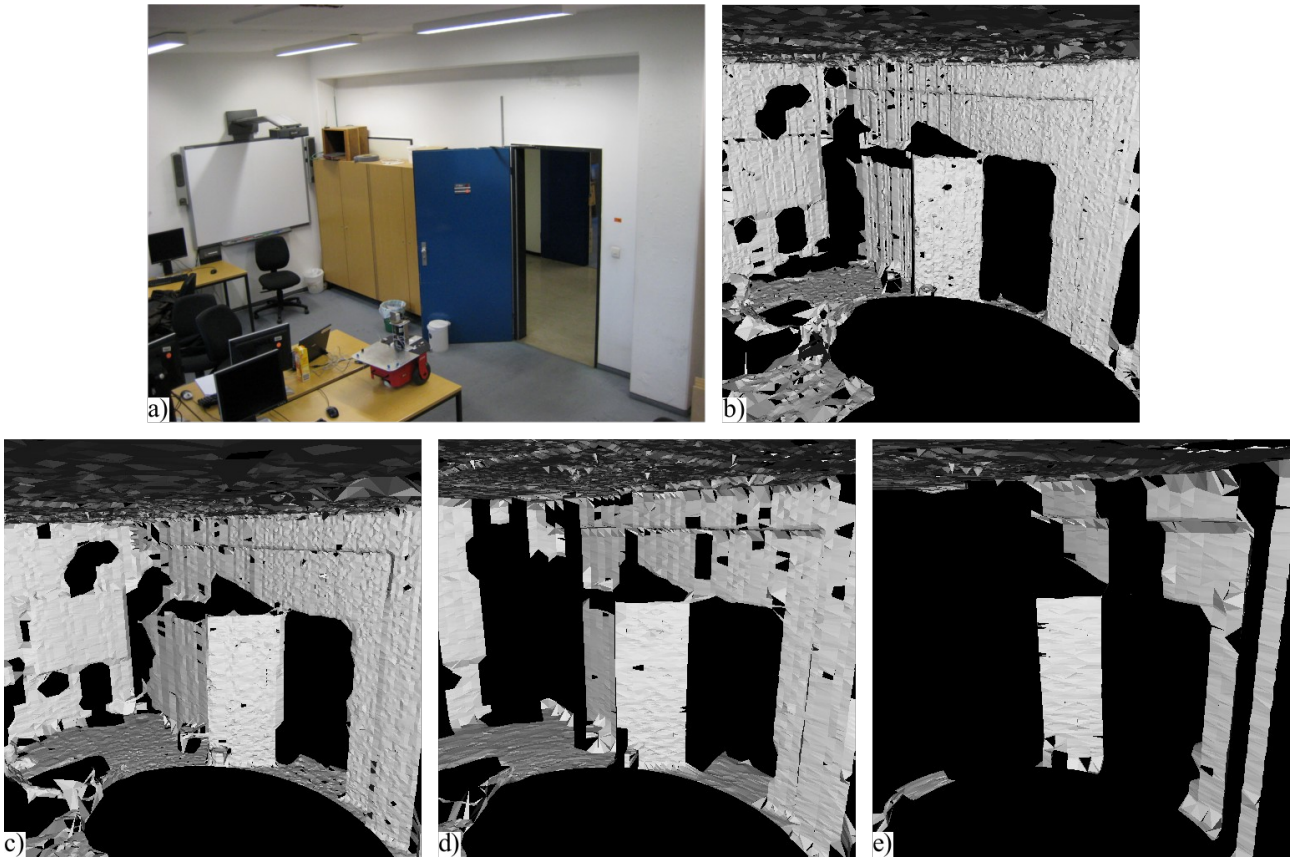


Abbildung 23: Darstellung der Messwerte mit hoher Reichweite nach Anwendung des Ball-Pivoting-Algorithmus
a) *Foto* b) *Schrittweite 0,45°*
c) *Schrittweite 0,9°* d) *Schrittweite 1,8°* e) *Schrittweite 3,6°*

Die Ergebnisse des Ball-Pivoting Algorithmus zeigen, dass die kartesischen Koordinaten der Messwerte mit hoher Auflösung durchaus geeignet sind Oberflächen zu rekonstruieren. Wände werden relativ gerade dargestellt und hervorstehende Gegenstände wie Tafeln sind auch zu erkennen. Bei geringer Auflösung werden die Flächen, deren Oberfläche nicht rekonstruiert werden konnte, größer. Da der Algorithmus immer dann versagt, wenn benachbarte Punkte sehr weit voneinander entfernt sind, ist es nicht verwunderlich, dass er weit entfernte Flächen am schlechtesten darstellen kann. Da es sich um ein mobiles System handelt, werden während der Bewegung noch weitere Messreihen durchgeführt, die für eine Verbesserung der Oberflächenrekonstruktion sorgen. Daher kann man das System mit niedrigster Auflösung zur Navigation und Kollisionsvermeidung verwenden, mit höheren Auflösungen ebenfalls zur Objekterkennung und zum Erstellen von virtuellen Realitäten.

5 Zusammenfassung Abschnitt III

Dieser Abschnitt enthielt die Ergebnisse der Messungen, die im Rahmen dieser Arbeit vorgenommen wurden.

Reaktionszeitmessungen mit der Verwendung des Linuxsystems haben ergeben, dass hinreichend schnell auf das Synchronisationssignal reagiert werden kann und der Motor genügend Zeit hat, seine nächste Position zu erreichen.

Die Messergebnisse einiger Messreihen wurden grafisch dargestellt, um einen Eindruck über deren Qualität zu vermitteln. Es zeigt sich, dass die Ergebnisse den gewünschten Anforderungen genügen und zur Lokalisierung und Kartenerstellung verwendet werden können.

IV Erweiterung des Systems auf den IBEO Lux

1 Einleitung Abschnitt IV

Um das System nicht nur in geschlossenen Räumen benutzen zu können, sondern auch im Freien nützliche Messdaten zu erhalten, ist der Laserscanner austauschbar. Man kann mit nur wenigen Handgriffen den leistungsstarken IBEO Lux Laserscanner montieren und so auf sehr große Entfernungen Messungen vornehmen.

In Kapitel 2 werden zunächst die Eckdaten und Eigenschaften des Laserscanners dargestellt, um eine grobe Vorstellung von dessen Leistung zu vermitteln.

Kapitel 3 widmet sich der Konstruktion der Laserscannerhalterung, während Kapitel 4 die Software beschreibt, die verwendet wurde, um Messungen mit dem IBEO Lux vorzunehmen.

In Kapitel 5 sind einige grafische Darstellungen und kurze Erklärungen von Messreihen mit dem IBEO Lux dargestellt.

Auch dieser Abschnitt endet mit einer Zusammenfassung in Kapitel 6.

2 Der Laserscanner

Der Laserscanner Lux von IBEO Automobile Sensor GmbH ist für den Automobilbau gedacht und befindet sich noch in der Entwicklung. Anwendungsbeispiele sind die Verwendung als Stauassistent, Kreuzungsassistent oder automatische Notbremsung zum Beispiel beim Fußgängerschutz [8]. Der TAMS-Gruppe wurde ein Lux von IBEO zur Verfügung gestellt und er wird zur Zeit ausgiebig getestet.

Abbildung 24: Der Laserscanner IBEO Lux [8]

Der IBEO-Laserscanner, siehe Abbildung 24, benutzt, genau wie der Hokuyo-Laserscanner, Infrarotlicht, jedoch mit einer größeren Wellenlänge (Wellenlänge λ=905 *nm*). Die Reichweite ist mit 0,3 *m* bis 200 *m* wesentlich höher als die des Hokuyo, daher eignet er sich auch besser für Vermessungen im Freien. Auch dieser Laserscanner liefert gewisse Messungenauigkeiten und

Fehler, allerdings gibt es aufgrund des Entwicklungsstatus noch keine genauen Angaben darüber. Messungen in der TAMS-Gruppe haben jedoch ergeben, dass relativ nahe Objekte näher und weiter entfernte Objekte etwas entfernter gemessen werden als sie sich wirklich befinden.

Dem Vorteil der sehr hohen Reichweite des Scanners stehen die größeren Abmessungen (85 *mm* x 128 *mm* x 93 *mm*) und das hohe Gewicht von etwa 1 *kg* gegenüber.

Die Kommunikation und der Datenversand finden entweder über die integrierte Ethernetschnittstelle oder den CAN-Bus statt. Es ist auch eine serielle Schnittstelle vorhanden, diese dient jedoch nur dem Debugging und sollte nicht im normalen Betrieb benutzt werden. Ebenso wie der URG04-LX besitzt der Laserscanner ein Synchronisationssignal, mit dessen Hilfe Messungen genau mit der Bewegung der Rotationsplattform abgepasst werden können. Ein Scan dauert mit etwa 80 *ms* nicht ganz so lange wie beim URG04-LX und es kann gleichzeitig in vier Ebenen gemessen werden, die etwa 1° auseinander liegen. Der Öffnungswinkel der Scanebenen beträgt dann 85°, bei einer Messung in nur zwei Ebenen 100° [8].

Der IBEO Lux benötigt für den Betrieb eine 12 *V* oder 24 *V* Spannungsquelle und hat eine Stromaufnahme von durchschnittlich 0,8 *A* bei einem Einschaltstrom von 2 *A*. Die Übertragung der Stromversorgung durch den Schleifring (5 *A* maximal) sollte also keine Schwierigkeiten bereiten.

3 Konstruktion

Ein Ziel dieser Arbeit war es, den Austausch des Laserscannermodells möglichst einfach zu gestalten. Um das zu erreichen wurde eine neue Plattform konstruiert, die den IBEO Lux trägt und mit der sich die Plattform, auf die der Hokuyo-Laserscanner montiert ist, mit nur wenigen Handgriffen austauschen lässt.

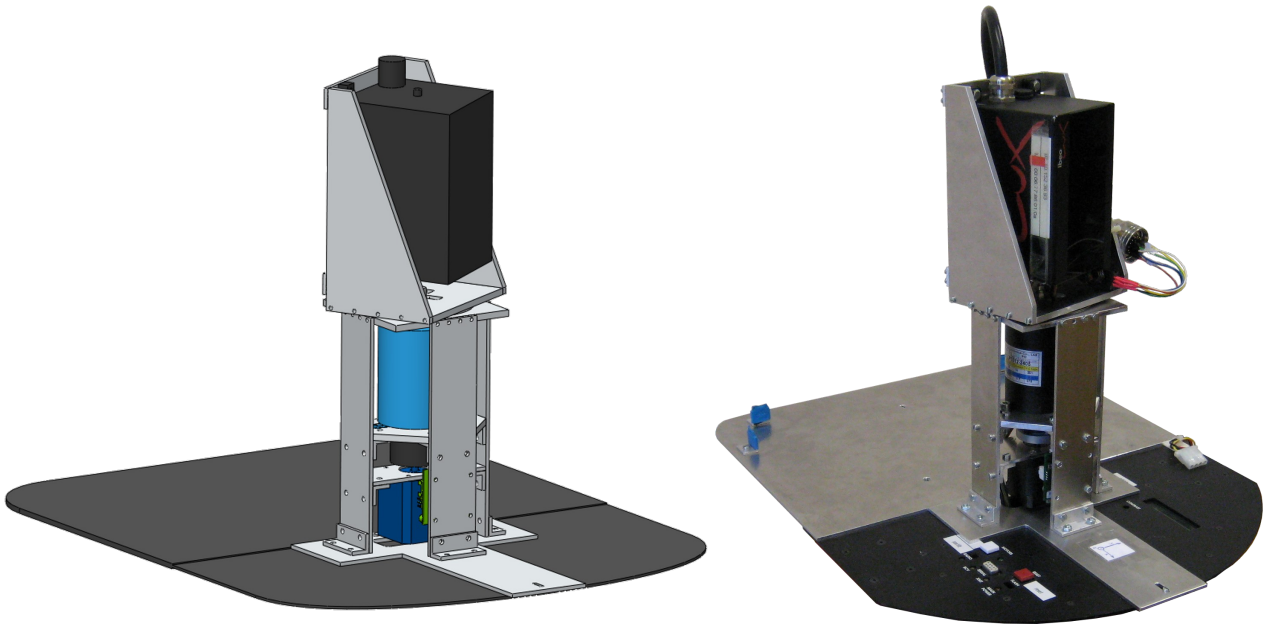
In die neue Plattform kann der Laserscanner hochkant eingeschraubt werden, sodass der Sensor-Koordinatenursprung [9] auf der Verlängerung der Motorachse liegt und sich während der Drehung nicht bewegt. Das vereinfacht spätere Koordinatentransformationen.

Die Konstruktion besteht, ebenso wie die für den Hokuyo-Laserscanner, aus einer Rotations-Ebene mit vier Öffnungen, durch die die vier Kabelbündel des Schleifrings durchgeführt werden. Auf beiden Seiten der Rotations-Ebene sind abgeschrägte Seitenstützen, an denen zwei Scannerhalterplatten befestigt sind. An diesen kann man den IBEO Lux mit vier Schrauben montieren.

Auch dieser Aufbau ist aus Aluminium gefertigt, um das Gewicht möglichst gering zu halten.

Die Kommunikation mit dem Laserscanner und die Datenübertragung erfolgt über eine Ethernetschnittstelle, die durch den Schleifring durchgeführt wird.

In Abbildung 25 ist die CAD-Zeichnung und ein Foto des endgültigen Aufbaus mit dem IBEO Lux zu sehen.



a) *CAD-Zeichnung* b) *Foto*

Abbildung 25: System mit montiertem IBEO Lux

4 Software

Nach Absprache wurde auf die Implementierung einer embedded Software, die die Daten auf dem Mikrocontrollerboard ausliest und verarbeitet, verzichtet. Stattdessen wurde der Aufbau getestet, indem die Messwerte des Laserscanners direkt mit Hilfe eines PCs ausgelesen wurden. Das Programm, das dafür verwendet wurde, basiert auf der von IBEO mitgelieferten API[7] und hat auch den Schrittmotor direkt angesprochen. Es wurde auch auf das Auslesen des Synchronisationssignals verzichtet, stattdessen wurde der Motor im Sekundentakt bewegt und eine Messung aus jedem Takt gespeichert.

Die Ergebnisse dieser Messungen sind im nächsten Kapitel dargestellt.

5 Messungen

Es wurde eine Messreihe durchgeführt, um die Funktionalität des Systems mit dem IBEO-Laserscanner zu zeigen. Die sich aus der Messreihe ergebene Punktewolke wurde, ebenso wie beim Hokuyo-Scanner, grafisch dargestellt, um einen Eindruck von dessen Qualität zu erhalten.

In Abbildung 26 a) sieht man ein Foto des Gebäudes, in dem sich die Arbeitsräume der TAMS-Gruppe befinden. Der Pioneer mit dem Aufbau ist direkt vor dem Gebäude im Blumenkübel zu erkennen. Abbildung 26 b) zeigt das Gebäude in der Darstellung als Punktewolke. Gemessen wurde in vier Ebenen und zwischen zwei Scans liegt ein Drehwinkel von 0,45°.

7 Application Programming Interface – eine Programmierschnittstelle

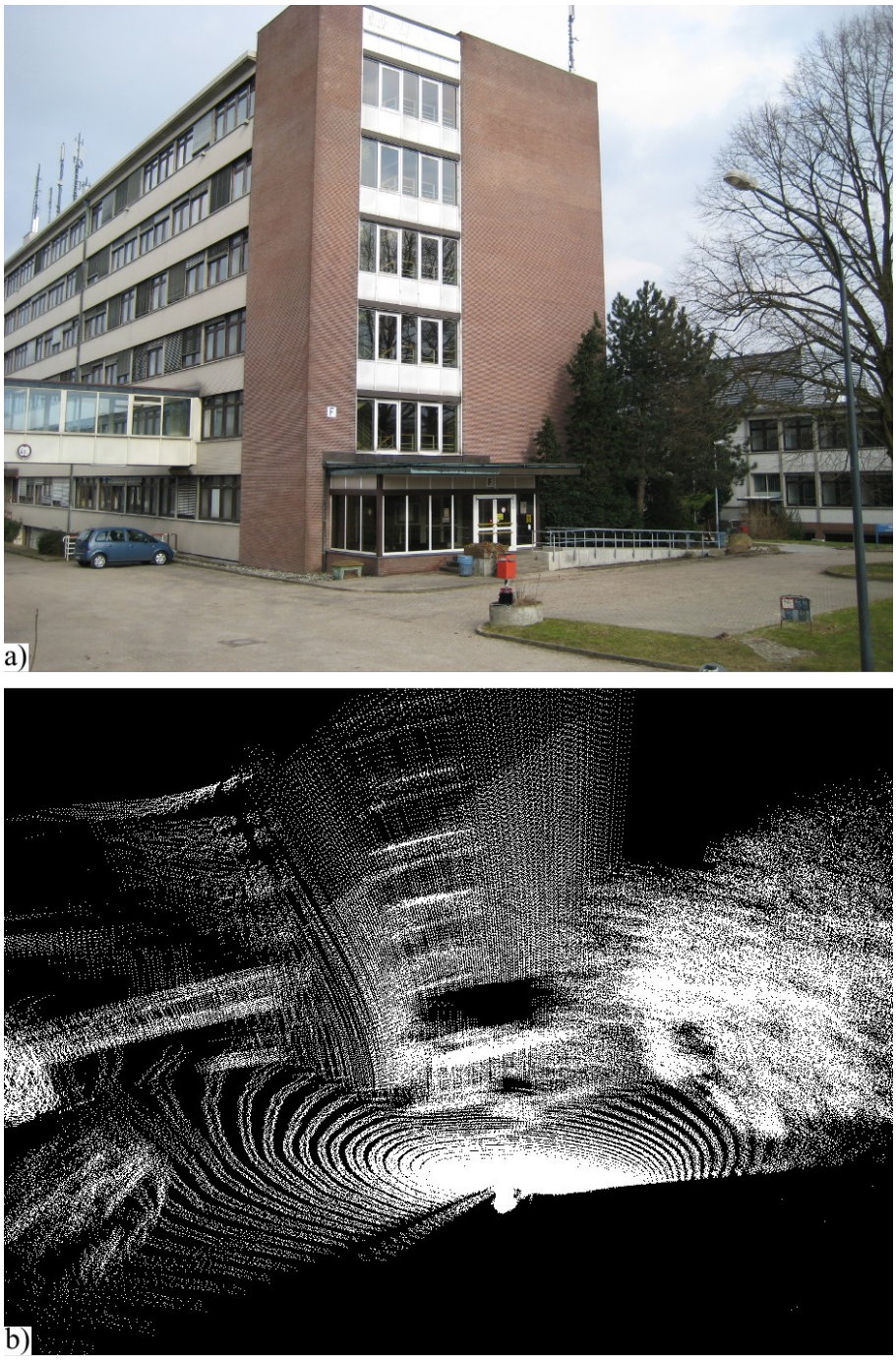


Abbildung 26: Messungen mit dem IBEO
a) *Foto* b) *Messpunktewolke*

Charakteristische Merkmale des Gebäudes, wie das Vordach und die von links hineinragende Fußgängerbrücke sind in der Punktewolke gut zu erkennen. Allerdings sieht man auch gleich, dass die Fassade des Gebäudes nicht gerade, sondern mit einer recht starken Krümmung dargestellt ist. Dies liegt höchstwahrscheinlich zum Einen an den in Kapitel 2 erwähnten Messungenauigkeiten und zum Anderen an der Umrechnung der Messdaten in kartesische Koordinaten. Diese erfolgt schon innerhalb des Laserscanners und wurde für die Messung nicht beeinflusst. Es besteht auch die

Möglichkeit, dass der Laserscanner an dem Rand seines Messbereichs sehr viel ungenauere Daten liefert als in der Mitte. Da der IBEO Lux für den Automobilbau entwickelt wird und hauptsächlich Objekte vor dem Fahrzeug zum Problem werden können, liegt diese Einschränkung des genauen Messbereichs nahe.

6 Zusammenfassung Abschnitt IV

In diesem Kapitel wurde gezeigt, dass der Umbau der Rotationsplattform auf den leistungsstärkeren IBEO Lux-Laserscanner möglich ist. Damit kann das System sowohl in Innenräumen als auch im Freien eingesetzt werden.

Es wurde die Konstruktion vorgestellt, die es ermöglicht, den IBEO Lux Laserscanner leicht zu montieren. Nach dem Prinzip der Konstruktion können ebenfalls andere Laserscanner oder Geräte leicht auf der Plattform angebracht und betrieben werden.

Es konnte gezeigt werden, dass die Qualität des Systems ausreicht, um Messungen im Freien auf größere Entfernungen aufzunehmen. Außerdem hat man durch diese erste Testmessung die ersten Schwierigkeiten aufgezeigt, die für zukünftige Arbeiten behoben werden müssen.

Zusammenfassung

Im Rahmen dieser Arbeit wurde ein System entwickelt, mit dem unter Verwendung verschiedener 2D-Laserscanner ein dreidimensionales Bild der Umgebung aufgenommen werden kann. Dies wird durch eine Rotation des Laserscanners um die vertikale Achse erreicht. Die Position und Ausrichtung des Scanners wurde so gewählt, dass die Umgebung mit einen halben Umdrehung komplett vermessen werden kann und sich die Roboterplattform, die das System trägt, während der gesamten Messung im toten Winkel des Laserscanners befindet. Die Messdaten werden direkt in dem System vorverarbeitet und erst für die spätere Verarbeitung, im besonderen mit SLAM-Algorithmen, an einen PC verschickt. Die grundlegenden Bedingungen der Aufgabenstellung an Portierbarkeit, Modularität, Erweiterbarkeit und Kostenbegrenzung wurden eingehalten und das System wurde ausgiebig auf Funktion getestet.

Das System wurde für einen Laserscanner mit relativ geringer Reichweite entwickelt und für den Einsatz auf einer Roboterplattform von ActiveMedia Robotics in geschlossenen Räumen optimiert. Die Messwerte werden direkt von dem System in kartesische und Polarkoordinaten umgerechnet und stehen für weitere Berechnungen, wie zum Beispiel der Kartenerstellung und der Entwicklung von Wegfindungsalgorithmen zur Verfügung.

Dank der Konstruktion aus Aluminium wurde ein geringes Gewicht erreicht und zusammen mit den relativ kleinen Abmessungen ist ein einfacher Umbau auf andere Roboterplattformen möglich. Durch Austausch der Grundplatte kann das System an nahezu jede Roboterplattform angepasst werden.

Die Modularität wurde mit dem Austausch des Laserscanners durch einen mit höherer Reichweite aufgezeigt. Messungen im Freien haben gezeigt, dass das System grundsätzlich auch für diesen Einsatz geeignet ist.

Das System bietet genügend Platz für Erweiterungen, so können weitere Laserscanner und Kameras an vielen Stellen an dem Aufbau montiert werden. Durch genügend Leitungen des Schleifrings wird gewährleistet, dass auch eine Vielzahl von Geräten auf der rotierenden Plattform angebracht werden können.

Das entstandene System wurde ausgiebig getestet und zeigt einige Schwächen, wenn es um Messungen mit großer Schrittweite geht, da dann viele Schrittverluste im Motor auftreten und die Scanebene bei Beginn des Scans nicht stillsteht. Bei höheren Auflösungen, das heißt kleineren Schrittweiten ist die Qualität der Messwerte und die Wiederholgenauigkeit sehr viel besser, auch wenn die Verarbeitung der Messdaten durch die größere Datenmenge etwas länger dauert.

Während der gesamten Arbeit wurden die Kosten im Auge behalten und so entstand ein System, das preislich sehr weit unter kommerziellen Systemen liegt und trotzdem eine Qualität bietet, die es auch für Forschungszwecke tauglich macht.

Ausblick

Da im Rahmen dieser Arbeit aus zeitlichen Gründen nicht sämtliche Visionen und Verbesserungen umgesetzt werden konnten, werden hier einige aufgezählt, um einen Überblick über die möglichen zukünftigen Fortschritte des Systems zu geben.

Der verwendete Schrittmotor besitzt keine Möglichkeit zur Positionsbestimmung und so können Schrittverluste und somit Positionierungsfehler nicht eindeutig erkannt werden. Abhilfe würde ein System zur Winkelmessung, zum Beispiel ein Inkrementalgeber, schaffen. Damit könnte man den Aufbau immer genau ausrichten und es wäre eine Möglichkeit gegeben, selbst bei großen Schrittverlusten die Referenzposition zu erreichen. Vor allem bei längeren Messfahrten wären Messungen dadurch zuverlässiger.

Eine Möglichkeit der Erweiterung des Systems ist, die Rotation mit konstanter Geschwindigkeit durchzuführen, anstatt vor jedem Scan den Motor zu stoppen. Das hat den Vorteil, dass weniger Schwingungen in dem System entstehen, was sich positiv auf die Messgenauigkeit und auf die Geräuschentwicklung des Systems auswirkt. Nachteil ist der Mehraufwand bei der Datenverarbeitung, da die Messwerte dann nicht mehr in einer Ebene liegen und die Rotation bei der Berechnung der kartesischen Koordinaten berücksichtigt werden muss.

Ebenfalls einen positiven Einfluss auf die Messergebnisse könnte die Verwendung eines Echtzeitbetriebssystems haben. Eine dadurch ermöglichte Garantie der Reaktionszeit würde zu konstanteren und zuverlässigeren Messwerten führen. Da außerdem eine sehr viel kürzere Reaktionszeit zu erwarten ist, hätte das System mehr Zeit die nächste Scanposition zu erreichen und es würden während der Messung weniger Schwingungen auftreten.

Um ein exakteres virtuelles Abbild der Realität zu erschaffen können Kamerabilder mit den Laserscanmesswerten überlagert werden. Auf diese Weise kann man die durchs Ball-Pivoting entstandenen Oberflächen mit Farben und Texturen füllen. Dabei kann man die Kamera (oder mehrere Kameras) entweder mitrotieren lassen oder fest an dem nicht rotierenden Aufbau montieren.

Die Geschwindigkeit des Systems kann erhöht werden, wenn man die vorverarbeiteten Messdaten anstatt per USB-Schnittstelle über die Ethernetschnittstelle überträgt. Wie in Kapitel I.2.3 erwähnt beschränkt sich die Übertragungsgeschwindigkeit des USB-Devices auf serielle Kommunikation, was zu den relativ langen Übertragungszeiten führt. Mit einer Ethernetverbindung könnten diese Daten sehr viel schneller übertragen werden.

Danksagung

Ich möchte mich herzlich bei Professor Teufel für die Hilfe bedanken, mit der er mir in Form von Ratschlägen, Kontaktpersonen und Literatur zur Seite stand, und auch dafür, dass er sich für meine Fragen immer Zeit genommen hat.

Außerdem danke ich Professor Zhang für seine Unterstützung, sein entgegengebrachtes Vertrauen und die reibungslose Zusammenarbeit, die dank ihm und Professor Teufel zwischen den beiden Universitäten stattfand.

Mein besonderer Dank gilt meinen Betreuern Denis Klimentjew und Houxiang Zhang, die immer Zeit für mich und meine Probleme und Fragen hatten und einen sehr großen Beitrag am Gelingen dieser Arbeit geleistet haben. Ich danke auch Herrn Florstett für die schnelle und einwandfreie Fertigung der Konstruktionselemente und der TAMS-Gruppe für den sehr angenehmen, freundschaftlichen Aufenthalt und interessante Einblicke in die Forschung.

Zu guter Letzt danke ich meiner Familie und meinen Freunden für ihre Unterstützung und Hilfsbereitschaft.

Ich freue mich, dass das System für weitere Arbeiten benutzt wird und wünsche jedem viel Spaß der sich weiter damit beschäftigt.

Literaturverzeichnis

[1] Hokuyo Automatic CO., LTD. URG Series Communication Protocol Specification, SCIP-Version2.0, 2006

[2] Hokuyo Automatic CO., LTD. Scanning Laser Range Finder URG-04LX – Specifications, 2005

[3] Trinamic Motion Control PANdrive PD-110-42 and TMCM-110-42, 2007

[4] Trinamic Motion Control TMCL – Reference and Programming Manual, 2008

[5] Embest Info&Tech Co., LTD. SBC6000X Hardware Manual, 2009

[6] Embest Info&Tech Co., LTD. SBC6000X Linux User Manual, 2009

[7] Embest Info&Tech Co., LTD. SBC6000X Linux Application Development Guide, 2009

[8] IBEO Automobile Sensor GmbH Datenblatt des IBEO Lux, 2008

[9] IBEO Automobile Sensor GmbH Betriebsanleitung des IBEO Lux, 2009

[10] Jan Axelson USB Complete – Third Edition, Lakeview Research - 2005

[11] Karim Yaghmour Building Embedded Linux Systems, O'Reilly - 2008

[12] Doug Abbott Linux for Embedded and Real-time Applications, Newnes - 2003

[13] H.-J. Weidemann Technische Mechanik, Teubner, 2002

[14] Delis, Christian Studienarbeit: Entwicklung einer Rotationsplattform für den Hokuyo URG-04LX, Universität Koblenz – Landau, 2006

[15] Karl Tyss Diplomarbeit: Untersuchung und Implementierung eines eingebetteten hybriden Echtzeitsystems auf Basis des Linux-Kernels, TUHH, 2009

[16] Aiwu Zhang Fast Continuous 360 Degree Color 3D Laser Scanner, Beijing - 2008

[17] Hartmut Surmann 6D SLAM – Preliminary Report on Closing the Loop in Six Dimensions, Fraunhofer Institute for Autonomous Intelligent Systems - 2007

[18] Kyeong-Hwan Lee Comparison of 2D laser scanners for sensing object distances, shapes and surface patterns, University of Florida, 2008

[19] Fausto Bernardini The Ball-Pivoting Algorithm for Surface Reconstruction, IEEE, 1999

[20] Michael Montemerlo FastSLAM: A Scalable Method for the Simultaneous Localization and Mapping Problem in Robotics, Springer, 2007

[21] http://www.torsten-horn.de/techdocs/ascii.html

Anhang

1 Bedienungsanleitung / User Manual

Software setup

Almost all the software you will need can be found in the software folder on the CD.
You will find additional information in *SBC6000X Linux User manual v1.2.pdf*

compile and load a program

Unpack *arm-linux-gcc-3.4.5-glibc-2.3.6-linux.tar.bz2* to the root directory.

To use the the cross-compiler execute
export PATH=/usr/crosstool/gcc-3.4.5-glibc-2.3.6/arm-linux/bin/:$PATH
This command has to be executed for every terminal window, where you want to use *arm-linux-gcc*, also for compiling the kernel.

now you can compile the diplom.c via the command
arm-linux-gcc filename.c -o filename -lpthread -lm

Most likely you will have to change the file rights: *chmod a+x+w+r filename*

To copy the output file to the microcontrollerboard, you need to setup a tftp-server on your PC with the transferfolder */tftpboot*.

Connect your PC with the microcontrollerboard via ethernet-cable.

Set your IP to 192.192.192.105, start the tftp-server and copy the desired file to */tftpboot*.

To use the linux on the board use a serial terminal program (for example cutecom)

To download the file to the board execute *tftp -g -r filename 192.192.192.105*

compile and load a new kernel

To compile the supplied kernel, you need to unpack *linux-2.6.24.tar.gz*.

Copy the file *sbc6000x_defconfig* to the *linux-2.6.24* folder and rename it to *.config*.

Replace the file /arch/arm/mach-at91/board-sbc9261.c with the one on the CD

Run make bzImage and wait a few minutes

Run *./mkimage* and a file named *uImage* is created.

Copy *uImage* to */tftpboot*.

To test the kernel start the board and send a random letter at bootup, to enter U-boot and execute *tftpboot 22000000 uImage* and wait till the kernel has been downloaded.

Then run *bootm 22000000*.

To install the kernel use the provided program (only for windows!) called *sam-ba* and follow the description written in *SBC6000X Linux User Manual v1.2.pdf*.

Run *./diplom* on the board to start the scanprogram

Run *menu-k* on the PC to start a simple program to control the scanner via USB.

Hardware Setup

1

- You will need the following tools:
  slotted screwdriver
  allen wrenches in the following sizes: 2 *mm*, 2.5 *mm*, 3 *mm*, 4 *mm*
  screw wrench: 8 *mm*

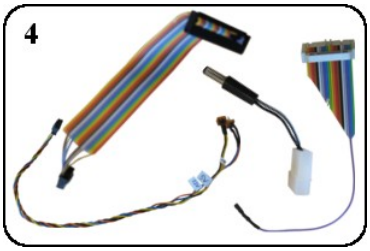
2

- Take off the sideplate and mount the serial-debug wire (male side) and the USB-data wire on the plate in the right openings. Lead the wires through the robot to the front and screw the sideplate back in position.

3

- Take off the frontplate of the robot
- Connect the wires to the SBC6000X and place the board (screwed to the board-mounting plate) in the front of the robot. The three holes in the plate fit to the three screws on the front side of the opened robot. Tighten the two nuts on the bottom screws to fix the board in position.

4

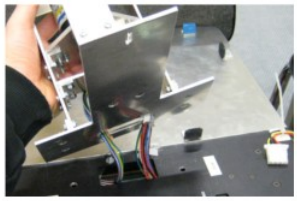


- Connect the serial-wire with the mark down and lead it under the board to a position on the right and behind the board.
- Connect the power-wire and lead it to the same position as the serial-wire.

5

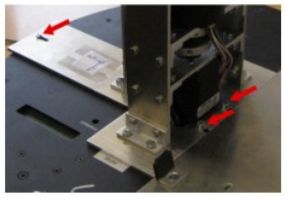
- Connect the 12 *V*-power-adapter to the battery connector and the board.
- Connect the serial wire with the fitting connector with six pins.

6



- Put the platform on the robot, leading all the wires through the opening in the middle. Make sure, all the wires, that are taped together go to the back, the others to the front of the robot. They should end up at the same position as the power and the serial-wires.

7



- Tighten the three screws fixing the platform.
- Put the needle-bearing on the platform.

Using the URG04-LX

- Connect the three-wire-power-connector with the fitting one labeled MOTOR and the two-wire-power-connector.
- Put the shaft on the rotating part of the Platform and mount the laserscanner on it.
- Mount the URG04-LX on the rotating plattform. Connect the scanner-cable.

8a

- Gently put the shaft with the rotating part on it through the slipring through-bore and lead the four connectors through the openings in the plate. Make sure the slipring is not tightened yet.
- Move the slipring with the shaft until it fits in the coupling.
- Press the rotating part gently on the bearing and tighten the couplings screw.

9a

- Turn the rotating part, to ensure its freedom of movement and then fix the slipring by tightening the screw.
- The four connectors from the top side of the slipring should now be connected to the inner connectors, mounted on the platform. Make sure the one labeled 1 is connected to the laserscanner-cable.
- Close the front plate of the robot.

Using the IBEO Lux

- Connect the power-splitter-cable to the three-wire-power-cable.
- Connect the two endings of the splitter with the wires labeled MOTOR and IBEO.
- Put the shaft on the rotating part of the Platform and mount the IBEO Lux on it.

8b

- Gently put the shaft with the rotating part on it through the slipring through-bore and lead the four connectors through the openings in the plate. Make sure the slipring is not tightened yet.
- Move the slipring with the shaft until it fits in the coupling.
- Press the rotating part gently on the bearing and tighten the couplings screw.

9b

- Turn the rotating part, to ensure its freedom of movement and then fix the slipring by tightening the screw.
- Connect the IBEO-scanner cable to the 26-pin-connector. The numbers on the wires and the connector should match. Connect the other side of the cable with the slipring-cable labeled 2.
- Connect the ethernet-adapter with the remaining connector with four pins inside the robot. Close the front plate of the robot.

2 Quellcode

Datei: *diplom.c*

Befehl zum Kompilieren: *arm-linux-gcc diplom.c -o diplom -lpthread -lm*

```
////////////////////////////////////////////////////////////////////////////
//      This program was written as part of a diploma thesis              //
//      at the technical university Hamburg-Harburg (TUHH)                //
//      in cooperation with the university of Hamburg (UHH)               //
//                                  |                                     //
//              author:             |       date:                         //
//                                  |                                     //
//              Breuer, Peter       |       march 2010                    //
//                                  |                                     //
////////////////////////////////////////////////////////////////////////////

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <time.h>
#include <unistd.h>
#include <termios.h>

#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <sys/select.h>

#include <linux/input.h>
#include <linux/fcntl.h>
#include <pthread.h>
#include <sys/signal.h>
#include <math.h>


#define MOTORPORT         "/dev/ttyS1"
#define SCANNERPORT       "/dev/ttyS2"
#define USBPORT           "/dev/ttyGS0"

//status:
        #define SCAN              1
        #define HOLD              0
        #define QUIT             -1      //QUIT only for debug, program should run without quit
        #define BEGIN             2
        #define END              -2
        #define PROCESS           3

        #define HIGH             1
        #define LOW              0

int     fd_scanner,
        fd_motor,
        fd_usb,
        status = HOLD,
        range = LOW,
        steps = 200,
        scancount = 0,
        cart[272400][3],
        spher[400][682];

long    cartsize,
        datacount;



char    command[9],
```

```
        scancommand[16],
        rawdata[880000];

FILE    *karthfile;


int open_port(char portname[11])
{
        int fd;
        fd = open(portname, O_RDWR | O_NOCTTY | O_NDELAY);
        if (fd == -1)
        {
                printf("Unable to open port %s\n", portname);
        }
        else
        {
                fcntl(fd, F_SETFL, 0);
                printf("%s opened\n", portname);
        }
        return (fd);
}

int close_port(int fd)
{
        close(fd);
        printf("port closed\n");
        return 0;
}

int configure_port(int fd, int baudrate)
{
        struct termios options;
//Get the current port-options
        tcgetattr(fd, &options);
//Set the baud rates
        switch(baudrate)
        {
                case 9600:
                        cfsetispeed(&options, B9600);
                        cfsetospeed(&options, B9600);
                break;
                case 19200:
                        cfsetispeed(&options, B19200);
                        cfsetospeed(&options, B19200);
                break;
                case 38400:
                        cfsetispeed(&options, B38400);
                        cfsetospeed(&options, B38400);
                break;
                case 57600:
                        cfsetispeed(&options, B57600);
                        cfsetospeed(&options, B57600);
                break;
                case 115200:
                        cfsetispeed(&options, B115200);
                        cfsetospeed(&options, B115200);
                break;
                case 500000:
                        cfsetispeed(&options, B500000);
                        cfsetospeed(&options, B500000);
                break;
                case 460800:
                        cfsetispeed(&options, B460800);
                        cfsetospeed(&options, B460800);
                break;
                default:
                        printf("unknown baudrate\n");
                        return -1;
        }
```

```
//set other options
        options.c_cflag  =  (options.c_cflag & ~CSIZE) | CS8;

        options.c_iflag &= ~(BRKINT | ICRNL | ISTRIP);
        options.c_iflag &= ~IXON;

        options.c_cflag &= ~PARENB;
        options.c_cflag &= ~CRTSCTS;
        options.c_cflag &= ~CSTOPB;

        options.c_oflag &= ~OPOST;

        options.c_lflag &= ~(ISIG | ICANON | ECHO);

        if (tcsetattr(fd, TCSANOW, &options)==0)
        {
                printf("Port configured (%dkbps)\n", baudrate);
        }
        return 0;
}

int compute_command(int stepsize)
{
        int i;
        unsigned char checksum;

        steps = 6400/stepsize;
        if (stepsize >=272)
        {
                printf("step too high\n");
                return -1;
        }
        printf("motor start\n");
//command: MVP-REL
        command[0]=0x01; //motor adress
        command[1]=0x04; //MVP
        command[2]=0x01; //REL
        command[3]=0x00;
        command[4]=0x00;
        command[5]=0x00;
        command[6]=0x00;
        command[7]=0x00;
        for(i=0; i<stepsize; i++)
        {
                command[7]=command[7]+0x01;
        }
        checksum = command[0];
        for(i=1; i<8; i++)
        {
                checksum += command[i];
        }
        command[8] = checksum;
        return 0;
}

int set_motor_velocity(int speed)
{
        char speedcommand[9];
        int n, i;
        unsigned char checksum;
        speedcommand[0]=0x01;     //motor adress
        speedcommand[1]=0x05;     //SAP
        speedcommand[2]=0x04;     //max. velocity
        speedcommand[3]=0x00;
        speedcommand[4]=0x00;
        speedcommand[5]=0x00;
        speedcommand[6]=0x00;
        speedcommand[7]=0x00;
        for(i=0; i<speed/256; i++)
```

```
        {
                speedcommand[6]=speedcommand[6]+0x01;
        }
        for(i=0; i<speed%256; i++)
        {
                speedcommand[7]=speedcommand[7]+0x01;
        }
        checksum = speedcommand[0];
        for(i=1; i<8; i++)
        {
                checksum += speedcommand[i];
        }
        speedcommand[8] = checksum;
        n = write(fd_motor, speedcommand, 9);
        if (n < 0)
        {
                fputs("write() of 9 bytes failed!\n", stderr);
                return -1;
        }
        sleep(1);
        return 0;
}
int set_motor_acceleration(int speed)
{
        char acccommand[9];
        int n, i;
        unsigned char checksum;
        acccommand[0]=0x01;       //motor adress
        acccommand[1]=0x05;       //SAP
        acccommand[2]=0x05;       //max. acceleration
        acccommand[3]=0x00;
        acccommand[4]=0x00;
        acccommand[5]=0x00;
        acccommand[6]=0x00;
        acccommand[7]=0x00;
        for(i=0; i<speed/256; i++)
        {
                acccommand[6]=acccommand[6]+0x01;
        }
        for(i=0; i<speed%256; i++)
        {
                acccommand[7]=acccommand[7]+0x01;
        }
        checksum = acccommand[0];
        for(i=1; i<8; i++)
        {
                checksum += acccommand[i];
        }
        acccommand[8] = checksum;
        n = write(fd_motor, acccommand, 9);
        if (n < 0)
        {
                fputs("write() of 9 bytes failed!\n", stderr);
                return -1;
        }
        sleep(1);
        return 0;
}

int move_motor() // moves motor left for a STEPSIZE
{
        int n;
        n = write(fd_motor, command, 9);
        if (n < 0)
        {
                fputs("write() of 9 bytes failed!\n", stderr);
                return -1;
        }
        return 0;
```

```
}

int send_data(int type)
{
        long i;
        short scan, value;
        char c[1], LB[1], HB[1], e[1], tmp[10];
        e[0]=0x00;
        c[0]=0x0A;
        switch (type)
        {
                case 1:  //send rawdata
                                printf("sending raw data\n");
                                for (i=0; i<datacount; i++)
                                {
                                        c[0] = rawdata[i];
                                        if (c[0] != 0x00)
                                                write(fd_usb, c, 1);
                                }
                                c[0]=0x0A;
                                write(fd_usb, e, 1);
                                printf("\ndata sent\n");
                break;
                case 2:  //send spherical data
                        printf("sending spherical data\n");
                        for (scan=0; scan<steps; scan++)
                        {
                                for (value=0; value<=681; value++)
                                {
                                        if (spher[scan][value] >= 0)
                                        {
                                                LB[0] = (spher[scan][value] % 80) + 0x28;
                                                HB[0] = (spher[scan][value] / 80) + 0x28;
                                        } else
                                        {
                                                LB[0] = ((-spher[scan][value]) % 80) + 0x28;
                                                HB[0] = ((-spher[scan][value]) / 80) + 0xA8; // + 0x28 + 0x80
                                        }
                                        write(fd_usb, LB, 1);
                                        write(fd_usb, HB, 1);
                                        write(fd_usb, c, 1);
                                }
                                write(fd_usb, c, 1);
                        }
                        write(fd_usb, e, 1);
                        printf("\ndata sent\n");
                break;
                case 3:  //send cartesian data
                        printf("sending cartesian data\n");
                        for (i=0; i<cartsize; i++)
                        {
                                if ((cart[i][0] != 0) && (cart[i][1] != 0) && (cart[i][2] != 0))
                                {
                                        if (cart[i][0] >= 0)
                                        {
                                                LB[0] = (cart[i][0] % 80) + 0x28;
                                                HB[0] = (cart[i][0] / 80) + 0x28;
                                        } else
                                        {
                                                LB[0] = ((-cart[i][0]) % 80) + 0x28;
                                                HB[0] = ((-cart[i][0]) / 80) + 0xA8; // + 0x28 + 0x80
                                        }
                                        write(fd_usb, HB, 1);
                                        write(fd_usb, LB, 1);
                                        if (cart[i][1] >= 0)
                                        {
                                                LB[0] = (cart[i][1] % 80) + 0x28;
                                                HB[0] = (cart[i][1] / 80) + 0x28;
                                        } else
```

```
                                        {
                                                LB[0] = ((-cart[i][1]) % 80) + 0x28;
                                                HB[0] = ((-cart[i][1]) / 80) + 0xA8; // + 0x28 + 0x80
                                        }
                                        write(fd_usb, HB, 1);
                                        write(fd_usb, LB, 1);
                                        if (cart[i][2] >= 0)
                                        {
                                                LB[0] = (cart[i][2] % 80) + 0x28;
                                                HB[0] = (cart[i][2] / 80) + 0x28;
                                        } else
                                        {
                                                LB[0] = ((-cart[i][2]) % 80) + 0x28;
                                                HB[0] = ((-cart[i][2]) / 80) + 0xA8; // + 0x28 + 0x80
                                        }
                                        write(fd_usb, HB, 1);
                                        write(fd_usb, LB, 1);
                                        write(fd_usb, c, 1);
                                }
                        }
                        write(fd_usb, e, 1);
                        printf("data sent\n");
                break;
        }
}

int clear_data()
{
        int a, b;
        for (a=0; a<880000; a++)
                rawdata[a]=0;

        for (a=0; a<400; a++)
                for (b=0; b<682; b++)
                        spher[a][b]=0;

        for (a=0; a<272400; a++)
                for (b=0; b<3; b++)
                        cart[a][b]=0;
}

int linesize(char line[200])
{
        int i, z=0;
        for (i=0; i < 200; i++)
        {
                if (line[i] == 0x0A)
                        return z;
                z++;
        }

}

int preprocess_data()     //compute the cartesian and spherical coordinates from the raw data
{
        long i=0;
        short a, z, scan, value, errors;
        double  xyz[3];
        status = PROCESS;
        printf("datacount%d\n", datacount);
//compute the spherical coordinates
        write(fd_usb, "preprocessing\n", 14);
        printf("preprocessing\n");
        errors=0;
        scan=-1;
        i=0;
        if (range == HIGH)
        {
                while (i<=datacount)
```

```
                {
                        if (    (rawdata[ i ]==scancommand[0])&&  //if command is found
                                (rawdata[i+1]==scancommand[1])&&
                                (rawdata[i+2]==scancommand[2])&&
                                (rawdata[i+3]==scancommand[3])&&
                                (rawdata[i+4]==scancommand[4])&&
                                (rawdata[i+5]==scancommand[5])    )
                        {
                                scan++;
                                value=0;
                                if (    (rawdata[i+ 91]!=0x0A)||//see if the linefeeds are in the right place
                                        (rawdata[i+ 157]!=0x0A)||
                                        (rawdata[i+ 223]!=0x0A)||
                                        (rawdata[i+ 289]!=0x0A)||
                                        (rawdata[i+ 355]!=0x0A)||
                                        (rawdata[i+ 421]!=0x0A)||
                                        (rawdata[i+ 487]!=0x0A)||
                                        (rawdata[i+ 553]!=0x0A)||
                                        (rawdata[i+ 619]!=0x0A)||
                                        (rawdata[i+ 685]!=0x0A)||
                                        (rawdata[i+ 751]!=0x0A)||
                                        (rawdata[i+ 817]!=0x0A)||
                                        (rawdata[i+ 883]!=0x0A)||
                                        (rawdata[i+ 949]!=0x0A)||
                                        (rawdata[i+1015]!=0x0A)||
                                        (rawdata[i+1081]!=0x0A)||
                                        (rawdata[i+1147]!=0x0A)||
                                        (rawdata[i+1213]!=0x0A)||
                                        (rawdata[i+1279]!=0x0A)||
                                        (rawdata[i+1345]!=0x0A)||
                                        (rawdata[i+1411]!=0x0A)||
                                        (rawdata[i+1477]!=0x0A)||
                                        (rawdata[i+1543]!=0x0A)||
                                        (rawdata[i+1609]!=0x0A)||
                                        (rawdata[i+1675]!=0x0A)||
                                        (rawdata[i+1741]!=0x0A)||
                                        (rawdata[i+1807]!=0x0A)||
                                        (rawdata[i+1873]!=0x0A)||
                                        (rawdata[i+1939]!=0x0A)||
                                        (rawdata[i+2005]!=0x0A)||
                                        (rawdata[i+2071]!=0x0A)||
                                        (rawdata[i+2135]!=0x0A)   )
                                {//if not every linefeed is in correct location,
                                 //the whole scan is filled with zero
                                        i+=2130;
                                        errors++;
                                        for (z = 0; z <= 681; z++)
                                                                spher[scan][z] = 0;
                                } else          //if every linefeed is in correct location
                                                //-> start processing
                                {
                                        a=1;
                                        for (z=26; z < 2133; z+=3)
                                        {
                                                if ((z == 24 + a*66)||(z+1 == 24 + a*66)||(z+2 == 24 + a*66))
                                                {
                                                        switch(a%3)
                                                        {
                                                                case 1:
                                                                        spher[scan][value] = 4096 * (int)
(rawdata[i+z] - 0x30) + 64 * (int)(rawdata[i+3+z] - 0x30) + (int)(rawdata[i+4+z] - 0x30);
                                                                        z+=2;
                                                                break;
                                                                case 2:
                                                                        spher[scan][value] = 4096 * (int)
(rawdata[i+z] - 0x30) + 64 * (int)(rawdata[i+1+z] - 0x30) + (int)(rawdata[i+4+z] - 0x30);
                                                                        z+=2;
                                                                break;
                                                                case 0:
```

```
                                                                spher[scan][value] = 4096 * (int)
(rawdata[i+2+z] - 0x30) + 64 * (int)(rawdata[i+3+z] - 0x30) + (int)(rawdata[i+4+z] - 0x30);
                                                                z+=2;
                                                        break;
                                                }
                                                a++;

                                        }
                                        else
                                                spher[scan][value] = 4096 * (int)(rawdata[i+z] - 0x30)
+ 64 * (int)(rawdata[i+1+z] - 0x30) + (int)(rawdata[i+2+z] - 0x30);
                                        if ((spher[scan][value] <= 20) || (spher[scan][value] >=
5600))
                                                spher[scan][value] = 0;
                                        value++;

                                }
                                i+=2130;
                        }
                } else
                {
                        i++;
                }
        }
} else   //range == LOW
{
        while (i<=datacount)
        {
                if (     (rawdata[ i ]==scancommand[0])&&  //if command is found
                        (rawdata[i+1]==scancommand[1])&&
                        (rawdata[i+2]==scancommand[2])&&
                        (rawdata[i+3]==scancommand[3])&&
                        (rawdata[i+4]==scancommand[4])&&
                        (rawdata[i+5]==scancommand[5])    )
                {
                        scan++;
                        value=0;
                        if (     (rawdata[i+ 91]!=0x0A)||//see if the linefeeds are in the right place
                                (rawdata[i+ 157]!=0x0A)||
                                (rawdata[i+ 223]!=0x0A)||
                                (rawdata[i+ 289]!=0x0A)||
                                (rawdata[i+ 355]!=0x0A)||
                                (rawdata[i+ 421]!=0x0A)||
                                (rawdata[i+ 487]!=0x0A)||
                                (rawdata[i+ 553]!=0x0A)||
                                (rawdata[i+ 619]!=0x0A)||
                                (rawdata[i+ 685]!=0x0A)||
                                (rawdata[i+ 751]!=0x0A)||
                                (rawdata[i+ 817]!=0x0A)||
                                (rawdata[i+ 883]!=0x0A)||
                                (rawdata[i+ 949]!=0x0A)||
                                (rawdata[i+1015]!=0x0A)||
                                (rawdata[i+1081]!=0x0A)||
                                (rawdata[i+1147]!=0x0A)||
                                (rawdata[i+1213]!=0x0A)||
                                (rawdata[i+1279]!=0x0A)||
                                (rawdata[i+1345]!=0x0A)||
                                (rawdata[i+1411]!=0x0A)||
                                (rawdata[i+1433]!=0x0A)   )
                        {//if not every linefeed is in correct location,
                         //the whole scan is filled with zero
                                i+=1400;
                                errors++;
                                for (z = 0; z <= 681; z++)
                                                        spher[scan][z] = 0;
                        } else   //if every linefeed is in correct location -> start processing
                        {
                                for (a=0; a<=20; a++)
                                        for (z=26+a*66; z < 89+a*66; z+=2)
```

```
                                                {
                                                        spher[scan][value] = 64 * (int)(rawdata[i+z] - 0x30) +
(int)(rawdata[i+1+z] - 0x30);
                                                        if ((spher[scan][value] <= 20) || (spher[scan][value]
>= 4094))
                                                                spher[scan][value] = 0;
                                                        value++;
                                                }
                                        for (z=1412; z < 1431; z+=2)      //last row
                                        {
                                                spher[scan][value] = 64 * (int)(rawdata[i+z] - 0x30) + (int)
(rawdata[i+1+z] - 0x30);
                                                if ((spher[scan][value] <= 20) || (spher[scan][value] >=
4094))
                                                        spher[scan][value] = 0;
                                                value++;
                                        }
                                        i+=1400;
                                }
                        } else
                        {
                                i++;
                        }
                }
        }
        printf("scans found: %03d\n", scan+1);
        printf("scans with errors: %03d\n", errors);
//compute cartesian coordinates
        double          pi1 = 168 * M_PI / 512,  //startangle in RAD change if gap occurs
                        pi2 = M_PI / 512,         //stepangle in RAD
                        sin1, cos1, sin2, cos2;
        cartsize=0;
        for (value = 0; value <= 681; value++)
        {
                if (spher[0][value] != 0)
                {
                        sin2 = sin(pi1 + value * pi2);
                        cos2 = cos(pi1 + value * pi2);
                        if (scancount % 2)                //check if scanner is in front or back position
                        {
                                xyz[0] =   0;    // X
                                xyz[1] = - cos1*sin2 * spher[0][value];   // Y
                                xyz[2] = -       cos2 * spher[0][value];   // Z
                        } else
                        {
                                xyz[0] = 0;      // X
                                xyz[1] =   cos1*sin2 * spher[0][value];   // Y
                                xyz[2] = -       cos2 * spher[0][value];   // Z
                        }
                        cart[cartsize][0] = (int)xyz[0];
                        cart[cartsize][1] = (int)xyz[1];
                        cart[cartsize][2] = (int)xyz[2];
                        cartsize++;
                }
        }
        for (scan = 1; scan < steps; scan++)
        {
                sin1 = sin(scan * M_PI / steps);
                cos1 = cos(scan * M_PI / steps);
                for (value = 0; value <= 681; value++)
                {
                        if (spher[scan][value] != 0)
                        {
                                sin2 = sin(pi1 + value * pi2);
                                cos2 = cos(pi1 + value * pi2);
                                if (scancount % 2)//check if scanner is in front or back position
                                {
                                        xyz[0] =    sin1*sin2 * spher[scan][value];        // X
                                        xyz[1] = -  cos1*sin2 * spher[scan][value];        // Y
```

```
                                        xyz[2] = -       cos2 * spher[scan][value];        // Z
                                } else
                                {
                                        xyz[0] = -  sin1*sin2 * spher[scan][value];        // X
                                        xyz[1] =    cos1*sin2 * spher[scan][value];        // Y
                                        xyz[2] = -       cos2 * spher[scan][value];        // Z
                                }
                                cart[cartsize][0] = (int)xyz[0];
                                cart[cartsize][1] = (int)xyz[1];
                                cart[cartsize][2] = (int)xyz[2];
                                cartsize++;
                        }
                }
        }
        printf("cartesian data processed\n");
        write(fd_usb, "ready\n", 6);
        status = HOLD;
}

void *read_usb(void *arg)//thread for listening to incoming command via USB and executing them
{
        int ret;
        char buffer[1], tmp[21];
        fd_set usb;
        FD_ZERO(&usb);
        FD_SET(fd_usb,&usb);
        while (status != QUIT)
        {
                ret = select(fd_usb+1, &usb, NULL, NULL, NULL);
                if (ret < 0)
                {
                        printf("error\n");
                }
                else if (ret == 0)
                {
                        break;
                }
                else
                {
                        read(fd_usb, buffer, 1);
                        if (buffer[0] == '1')
                        {
                                if (status == HOLD)
                                {
                                        set_motor_acceleration(2047);
                                        set_motor_velocity(2047);
                                        compute_command(16);
                                        write(fd_usb, "stepsize 16\n", 12);
                                        printf("stepsize = 16\n");
                                } else
                                {
                                        write(fd_usb, "error\n", 6);
                                        printf("couldn't change stepsize\n");
                                }
                        }
                        if (buffer[0] == '2')
                        {
                                if (status == HOLD)
                                {
                                        set_motor_acceleration(2047);
                                        set_motor_velocity(2047);
                                        compute_command(32);
                                        write(fd_usb, "stepsize 32\n", 12);
                                        printf("stepsize = 32\n");
                                } else
                                {
                                        write(fd_usb, "error\n", 6);
                                        printf("couldn't change stepsize\n");
                                }
```

```
}
if (buffer[0] == '3')
{
        if (status == HOLD)
        {
                set_motor_acceleration(2047);
                set_motor_velocity(2047);
                compute_command(64);
                write(fd_usb, "stepsize 64\n", 12);
                printf("stepsize = 64\n");
        } else
        {
                write(fd_usb, "error\n", 6);
                printf("couldn't change stepsize\n");
        }
}
if (buffer[0] == '4')
{
        if (status == HOLD)
        {
                set_motor_acceleration(100);
                set_motor_velocity(100);
                compute_command(128);
                write(fd_usb, "stepsize 128\n", 13);
                printf("stepsize = 128\n");
        } else
        {
                write(fd_usb, "error\n", 6);
                printf("couldn't change stepsize\n");
        }
}
if (buffer[0] == 's')
{
        clear_data();
        write(fd_usb, "scanning\n", 9);
        printf("start scanning\n");
        status = BEGIN;
        datacount = 0;
}
if (buffer[0] == 'h')
{
        if (status == HOLD)
                if (range == HIGH)
                {
                        range = LOW;
                        setSCIP2();
                        sprintf(scancommand, "MS0044072501000\n");
                        write(fd_scanner, scancommand, 16);
                        write(fd_usb, "range set to LOW\n", 17);
                        printf("range set to LOW\n");
                } else
                {
                        range = HIGH;
                        setSCIP2();
                        sprintf(scancommand, "MD0044072501000\n");
                        write(fd_scanner, scancommand, 16);
                        write(fd_usb, "range set to HIGH\n", 18);
                        printf("range set to HIGH\n");
                }
        else
        {
                write(fd_usb, "couldn't set range\n", 19);
                printf("couldn't set range\n");
        }
}
if (buffer[0] == 'v')
{
        if (scancount == 1)
                sprintf(tmp, "%04d scan completed\n", scancount);
```

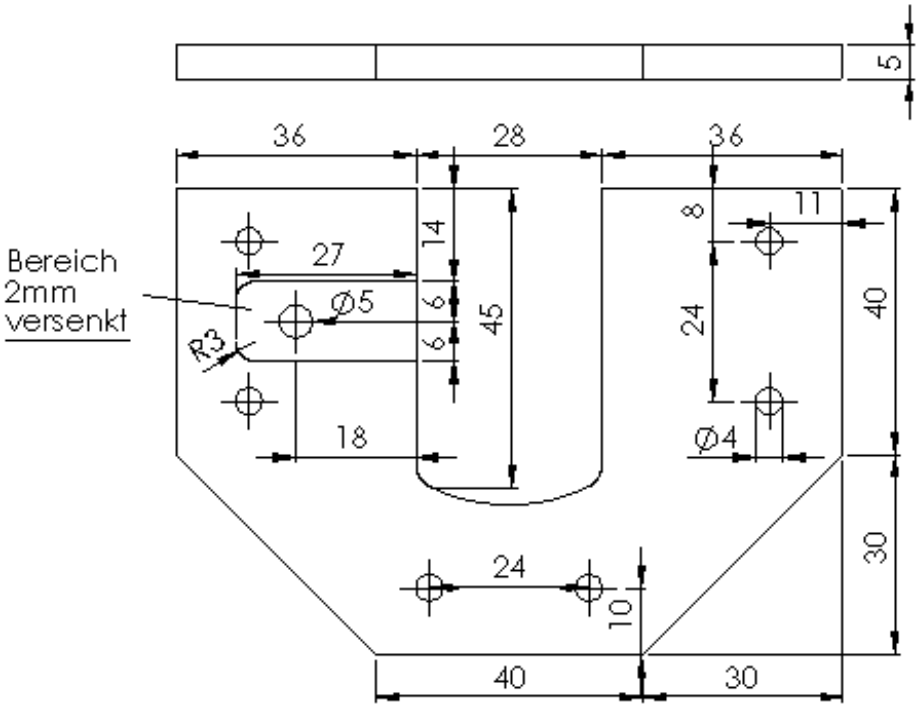
```
                                else
                                        sprintf(tmp, "%04d scans completed\n", scancount);
                                write(fd_usb, tmp, 21);
                                if (status == SCAN)
                                        write(fd_usb, "status: scanning\n", 17);
                                if (status == HOLD)
                                        write(fd_usb, "status: pausing\n", 16);
                                if (status == PROCESS)
                                        write(fd_usb, "status: processing data\n", 24);
                        }
                        if (buffer[0] == 'r')     //send rawdata
                        {
                                if (status == SCAN)
                                {
                                        write(fd_usb, "scanning in progress\n", 21);
                                } else if (status == PROCESS)
                                {
                                        write(fd_usb, "processing in progress\n", 23);
                                } else
                                {
                                        send_data(1);
                                }
                        }
                        if (buffer[0] == 't')     //send spherical data
                        {
                                if (status == SCAN)
                                {
                                        write(fd_usb, "scanning in progress\n", 21);
                                } else if (status == PROCESS)
                                {
                                        write(fd_usb, "processing in progress\n", 23);
                                } else
                                {
                                        send_data(2);
                                }
                        }
                        if (buffer[0] == 'z')     //send cartesian data
                        {
                                if (status == SCAN)
                                {
                                        write(fd_usb, "scanning in progress\n", 21);
                                } else if (status == PROCESS)
                                {
                                        write(fd_usb, "processing in progress\n", 23);
                                } else
                                {
                                        send_data(3);
                                }
                        }
                        if (buffer[0] == 'q')
                        {
                                write(fd_usb, "quit\n", 5);
                                printf("quit scanning\n");
                                status = HOLD;
                        }
                        if (buffer[0] == 'Q')
                        {
                                write(fd_usb, "quitting program\n", 17);
                                printf("quitting program\n");
                                status = QUIT;
                        }
                }
        }
}

void *read_gpio(void *arg)
{
        int ret, count=0, i, fd_gpio, fd_gpiobus;
        fd_set gpio;
```

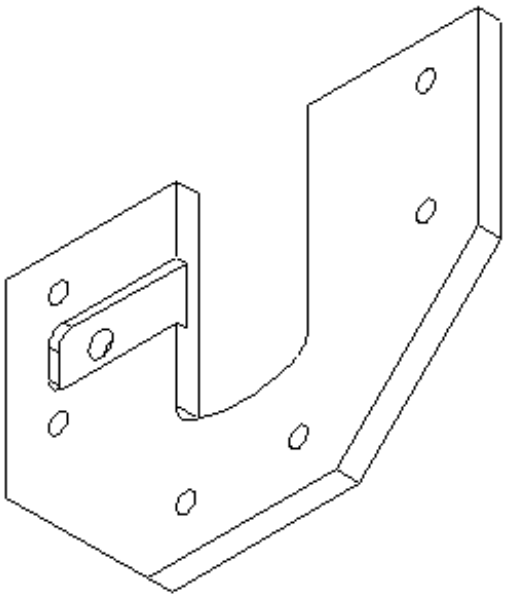
```
struct input_event ev;

fd_gpio = open("/dev/input/event2",O_RDONLY);
if(fd_gpio<0)
{
        perror("can not open interrupt-device");
}
FD_ZERO(&gpio);
FD_SET(fd_gpio,&gpio);
while(status != QUIT)
{
        ret = select(fd_gpio+1, &gpio, NULL, NULL, NULL);  //infinit loop
        if(ret < 0)
                perror("select");
        else if(ret == 0)
        {
                break;
        }
        else
        {
                if(FD_ISSET(fd_gpio, &gpio))
                {
                        if(read(fd_gpio, (char *)&ev, sizeof(ev))>0)
                        {
                                switch(ev.type)
                                {
                                        case EV_KEY:
                                                switch(ev.code)
                                                {
                                                        case BTN_0:/* Event for Button 0 (ignore)*/
                                                                break;
                                                        case BTN_1:/* Event for Button 1 (ignore)*/
                                                                break;
                                                        case BTN_2:/* Event for GPIO */
                                                                if (ev.value==1)
                                                                {
                                                                        if (status == SCAN)
                                                                        {
                                                                                count++;
                                                                                move_motor();
                                                                                if (count == steps)
                                                                                {
                                                                                  status = END;
                                                                                  count = 0;
                                                                                  scancount++;
                                                                                  preprocess_data();
                                                                                }
                                                                        }
                                                                        if (status == BEGIN)
                                                                                status = SCAN;
                                                                        if (status == END)
                                                                                status = HOLD;
                                                                }
                                                                FD_ZERO(&gpio);
                                                                FD_SET(fd_gpio,&gpio);
                                                                break;
                                                        default:
                                                                break;
                                                }
                                                break;
                                        default:
                                                break;
                                }
                        }
                }
        }
}
close(fd_gpio);
FD_CLR(fd_gpio,&gpio);
```

```
}

int setSCIP2()
{
        int n;
// reset scanner
        write(fd_scanner, "RS\n",3);
        printf("Scanner reset\n");
// set SCIP2-mode
        n = write(fd_scanner, "SCIP2.0\n", 8);
        if (n < 0)
        {
                fputs("couldn't set SCIP2.0!\n", stderr);
                return -1;
        }
        printf("set in SCIP2.0 mode\n");
        sleep(1);
        return 0;
}

int disable_scanner()
{
        char tmp[3];
        sprintf(tmp, "QT\n");
        write(fd_scanner, tmp, 3);
}
int enable_scanner()
{
        char tmp[3];
        sprintf(tmp, "BM\n");
        write(fd_scanner, tmp, 3);
}

int startscan()
{
        char tmp[16], timestamp[4];
        char buffer[1];
        int messw[5000] = {0}, z, i, ret, j;

        fd_set scanner;
        FD_ZERO(&scanner);
        FD_SET(fd_scanner,&scanner);
// create scancommand
        if (range == LOW)
        {
                sprintf(scancommand, "MS0044072501000\n");
        }
        if (range == HIGH)
        {
                sprintf(scancommand, "MD0044072501000\n");
        }
// send command
        write(fd_scanner, scancommand, 16);       //-> Laser turns on!! (in the scanning area)
        printf("ready\n");
        write(fd_usb, "ready\n", 6);
// receive data
        while (status != QUIT)
        {
                ret = select(fd_scanner+1, &scanner, NULL, NULL, NULL);    //infinit loop
                        if (ret < 0)
                        {
                                printf("error\n");
                        }
                        else if (ret == 0)
                        {
                                break;
                        }
                        else
```

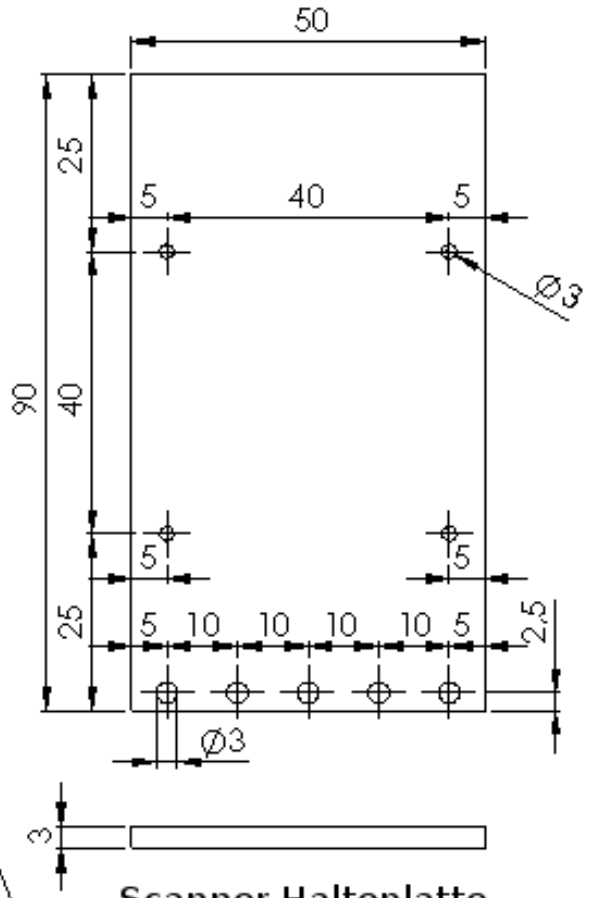
```
                        {
                                read(fd_scanner, buffer, 1);              //read all the data
                                if ((status == SCAN) || (status == END))
                                {
                                        rawdata[datacount]=buffer[0];     //write the data in rawdata
                                        datacount++;
                                }
                        }
        }
        return 0;
}

int main()
{
        pthread_t gpiothread, usbreadthread;

        fd_motor   = open_port(MOTORPORT);
        fd_scanner = open_port(SCANNERPORT);
        fd_usb     = open_port(USBPORT);
        if ((fd_motor == -1) || (fd_scanner == -1))
        {
                printf("couldn't open the serial ports, please check them.\n");
                return -1;
        }
        while (fd_usb == -1)
        {
                fd_usb = open_port(USBPORT);
                sleep(1);
        }
        configure_port(fd_motor, 57600);
        configure_port(fd_scanner, 500000);
        configure_port(fd_usb, 460800);

        set_motor_acceleration(2047);     //motor default settings, stepsize 64
        set_motor_velocity(2047);
        compute_command(64);

        setSCIP2();
        if ( pthread_create( &gpiothread, NULL, read_gpio, NULL) )
        {
                printf("error creating gpiothread.");
                abort();
        }
        if ( pthread_create( &usbreadthread, NULL, read_usb, NULL) )
        {
                printf("error creating usbthread.");
                abort();
        }
////////////////////////////////////PARALLEL TO THE THREAD/////////////////////////////////////////
        startscan();
////////////////////////////////////PARALLEL TO THE THREAD/////////////////////////////////////////
        if ( pthread_join ( usbreadthread, NULL ) )
        {
                printf("error joining usbthread.");
                abort();
        }
        if ( pthread_join ( gpiothread, NULL ) )
        {
                printf("error joining gpiothread.");
                abort();
        }
        return 0;
}
```

3 Konstruktionszeichnungen

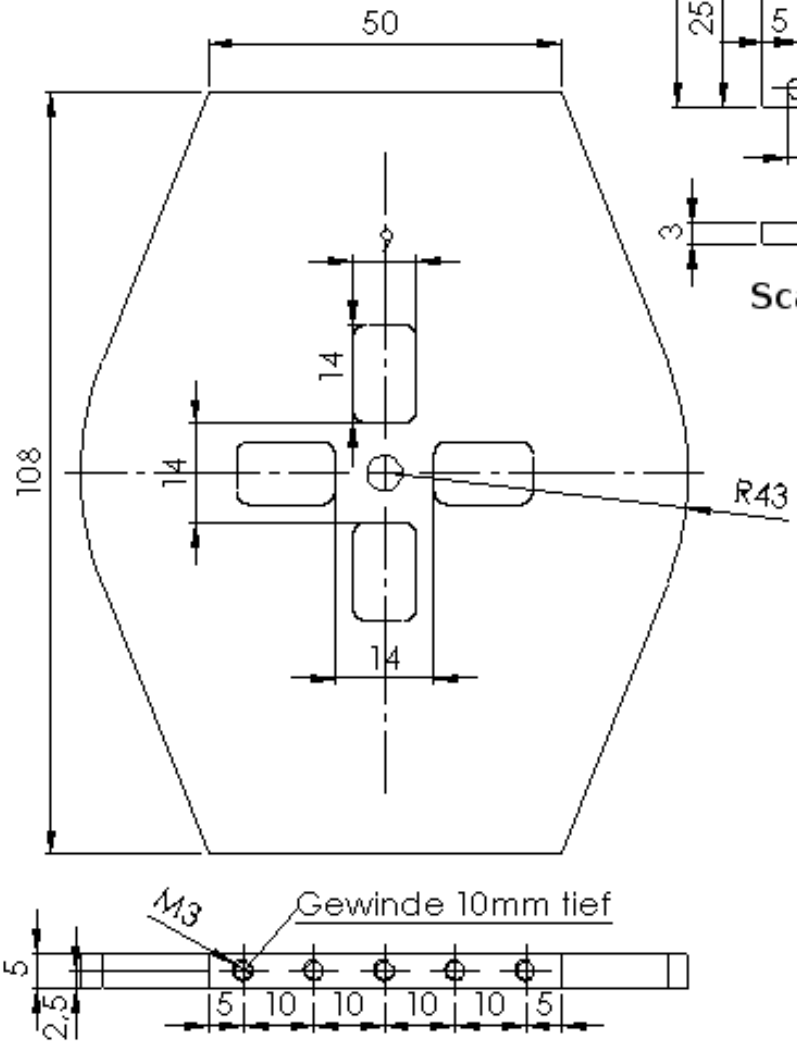




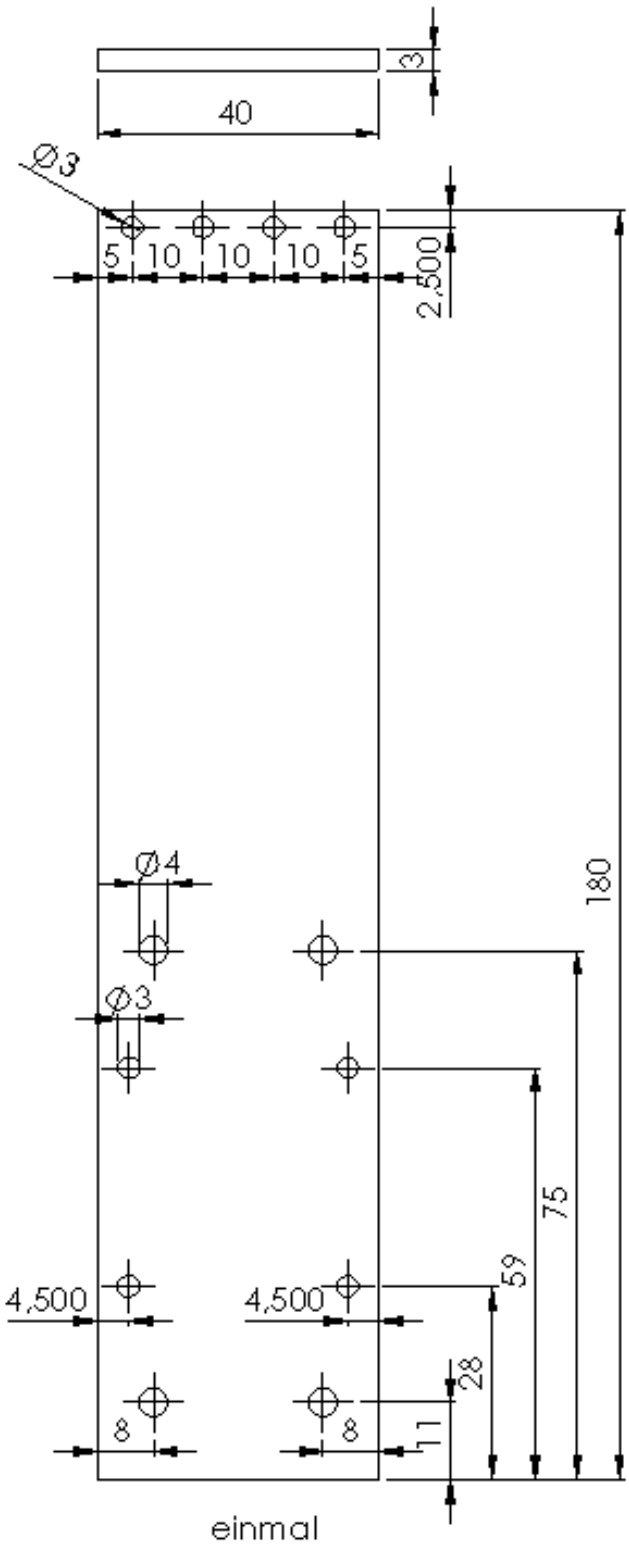
Schleifring-Ebene



Scanner-Halteplatte
- Hokuyo

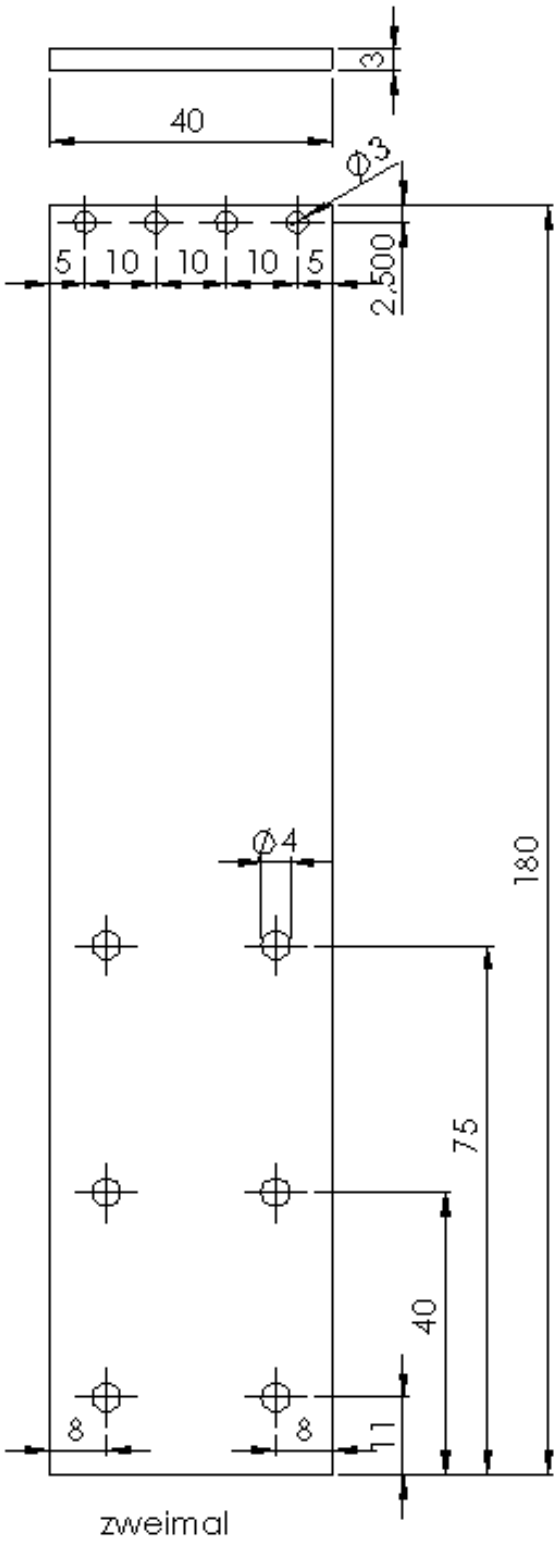


Rotations-Ebene - Hokuyo



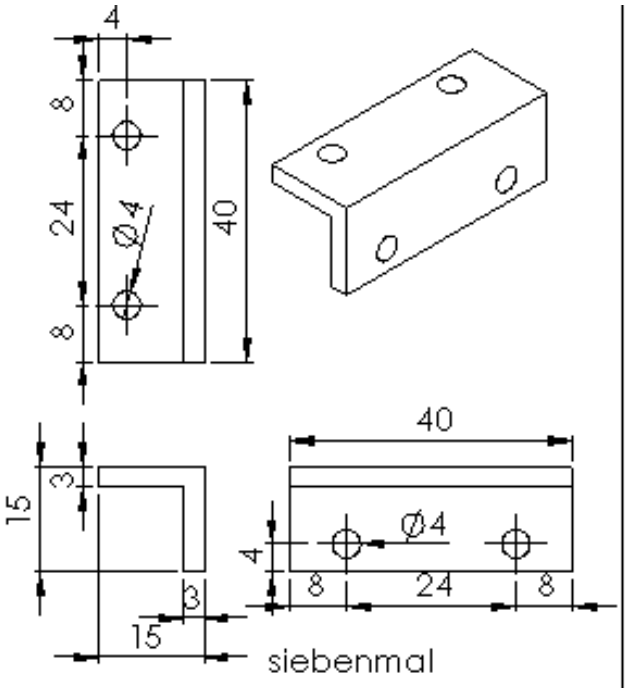
einmal

Stütze - Mitte

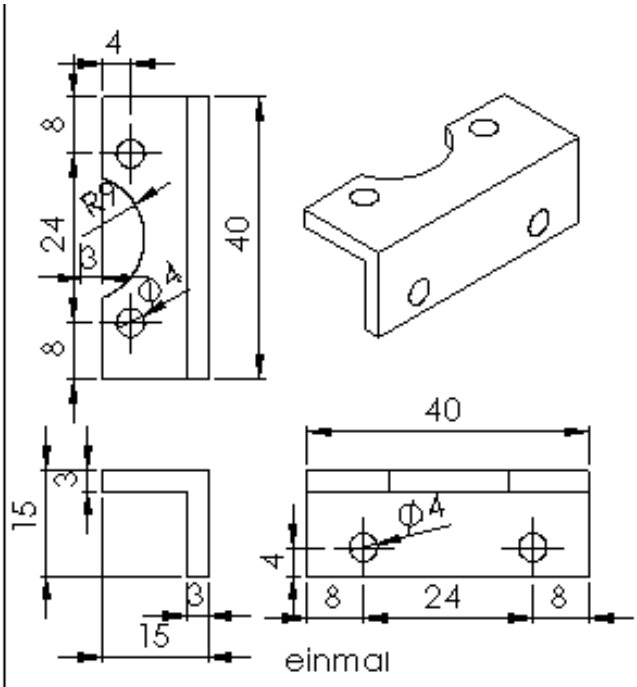


zweimal

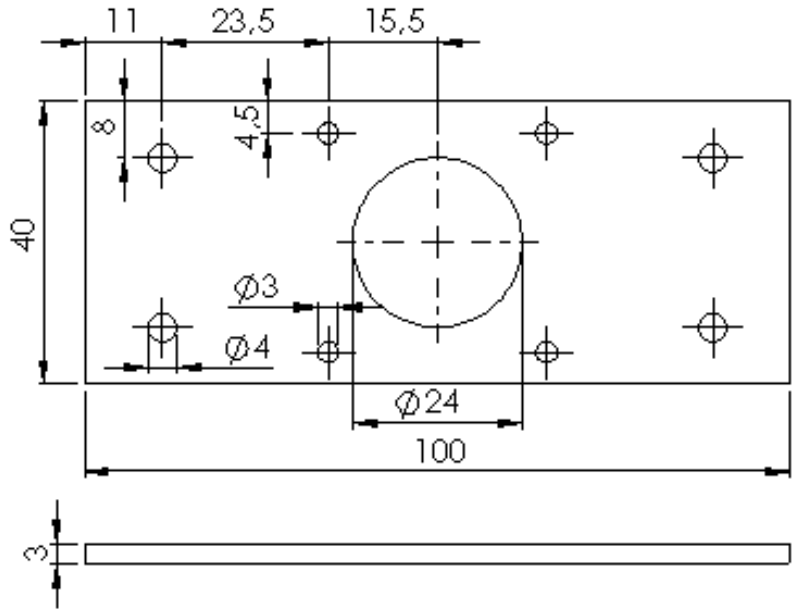
Stütze - Seite



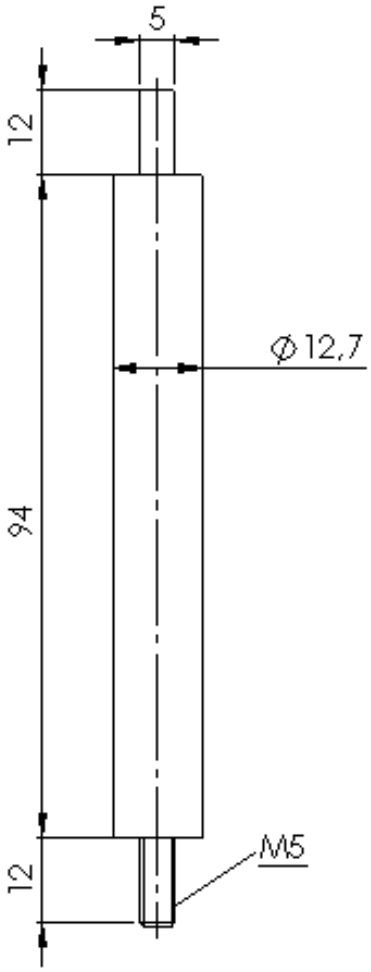
Winkel



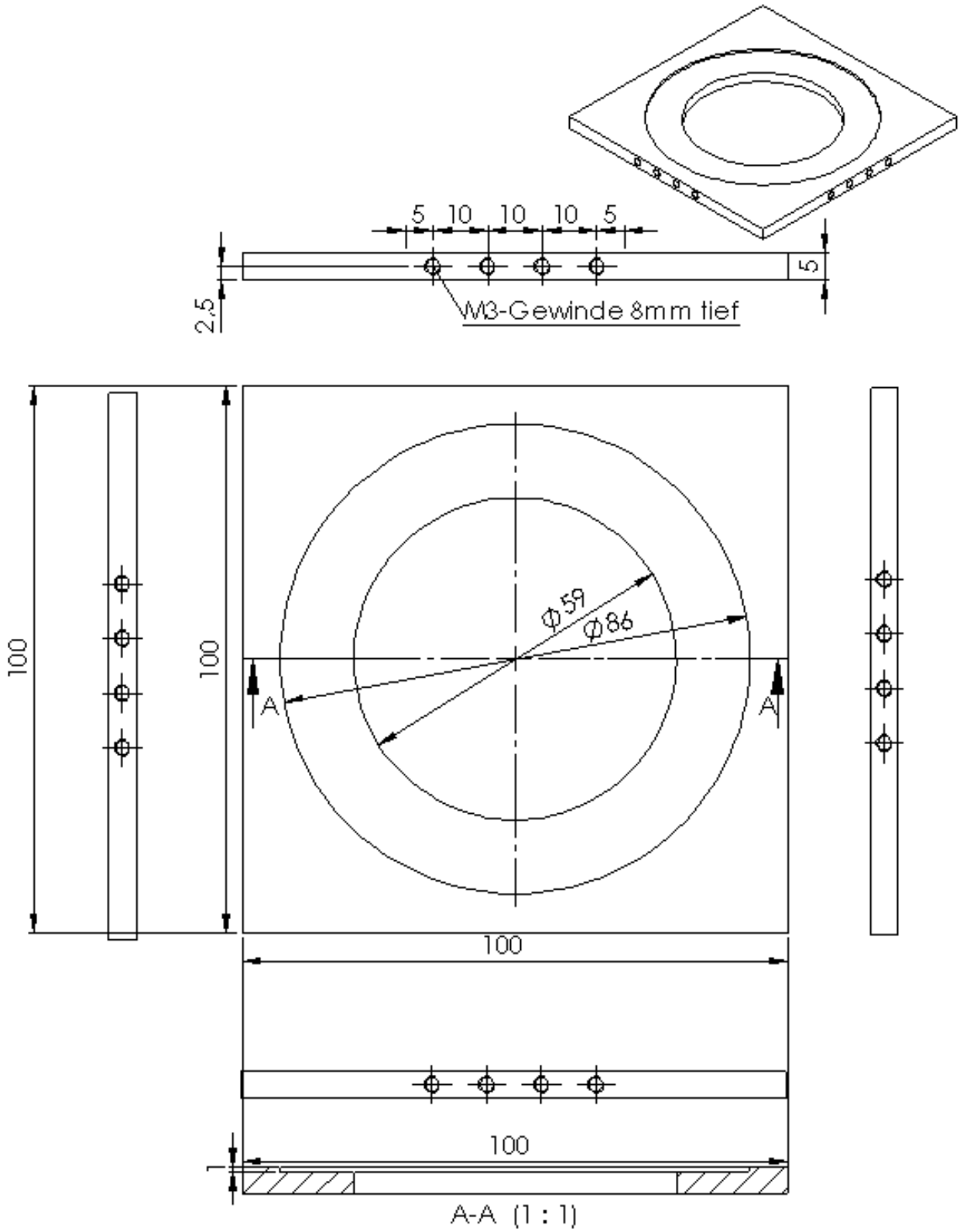
Winkel-Schleifring



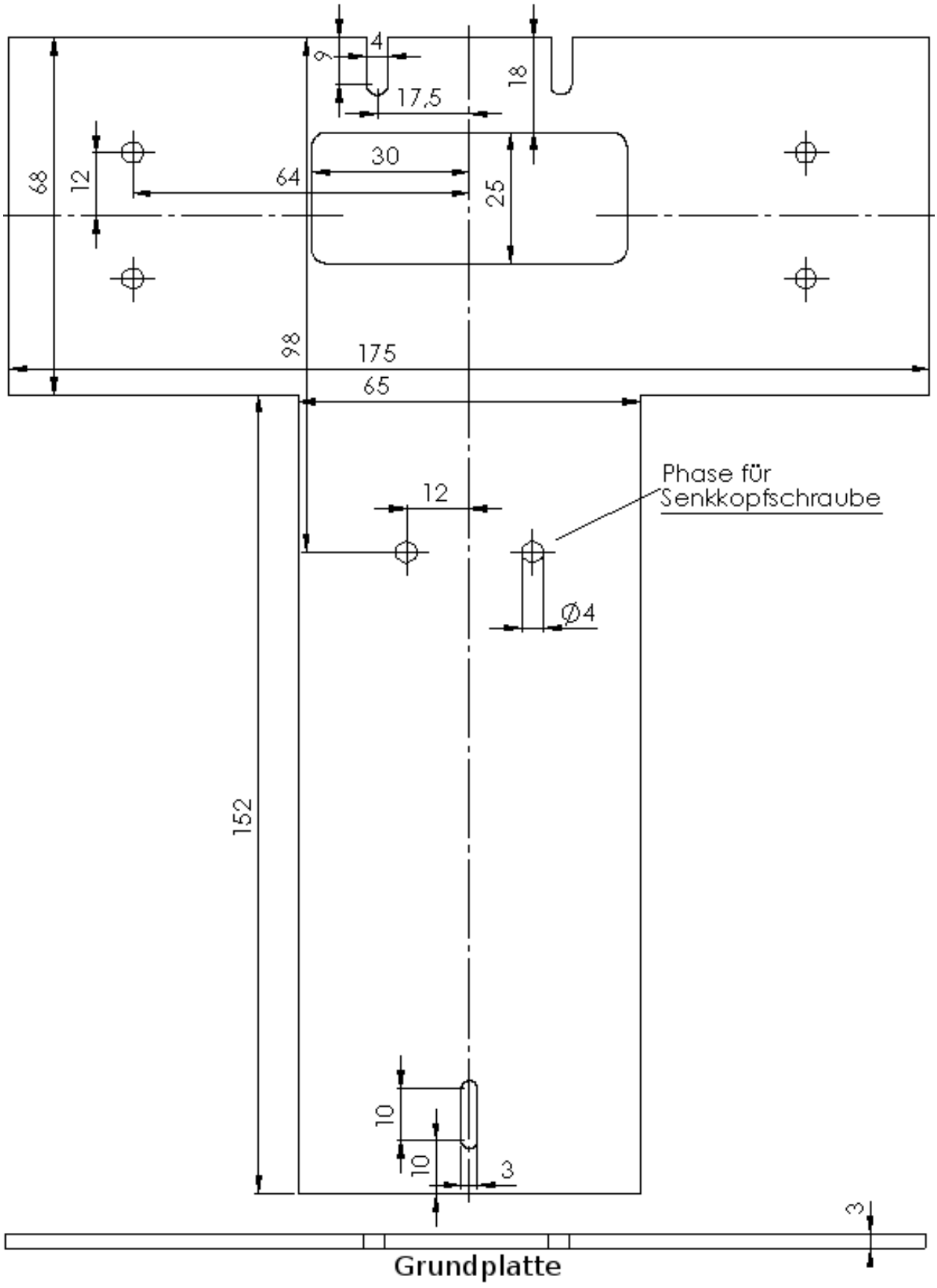
Motor-Ebene

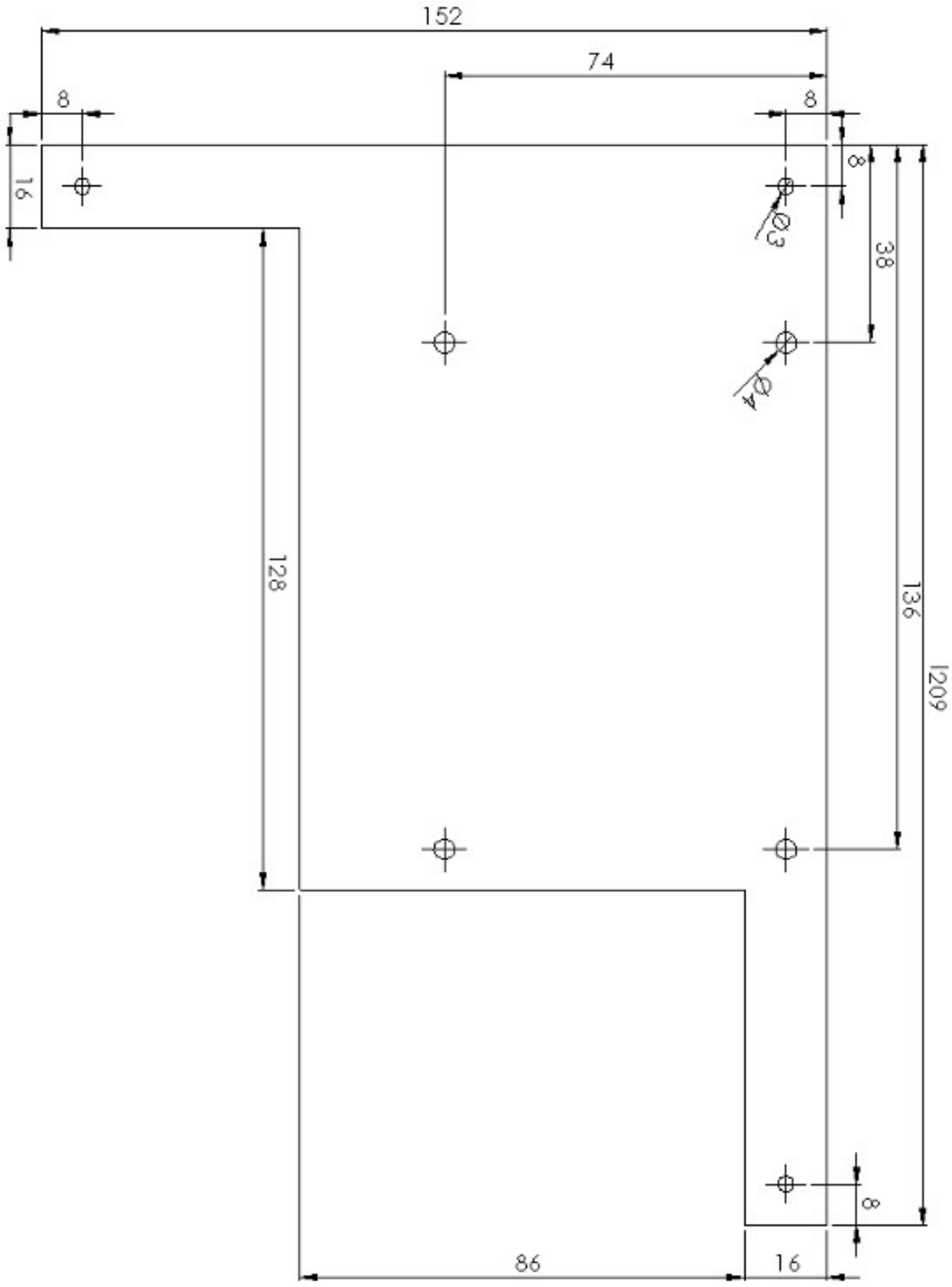


Welle

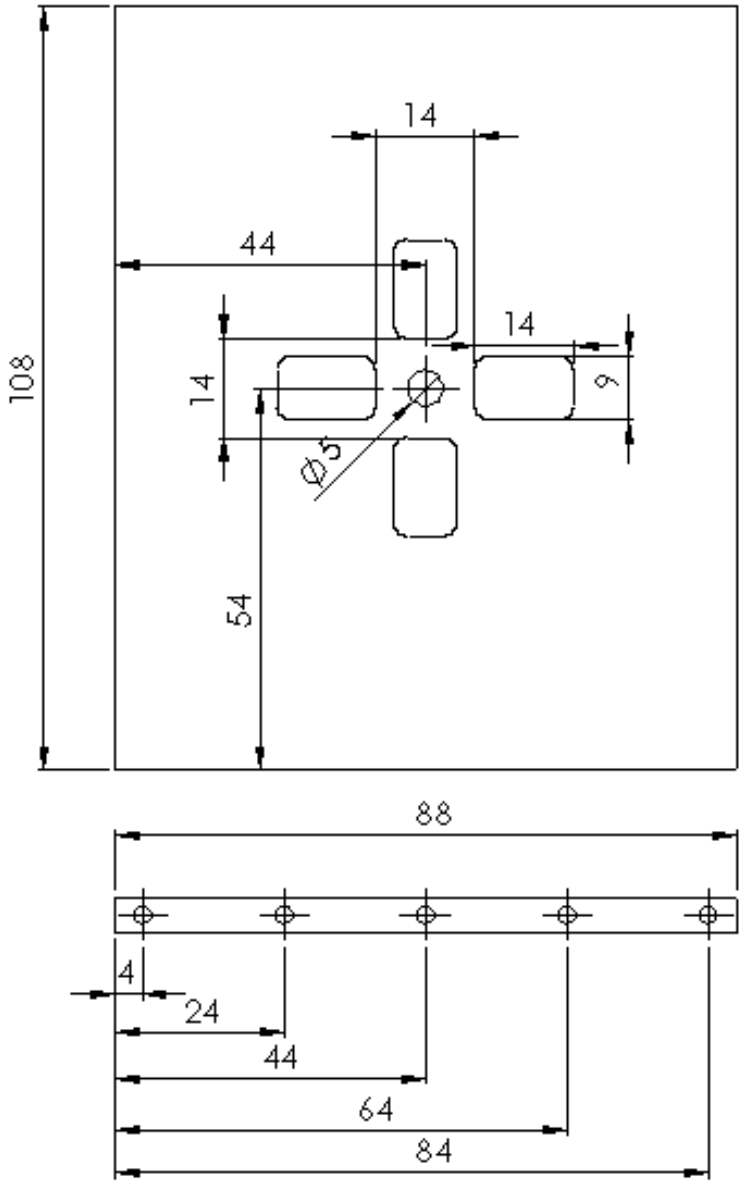

5 10 10 10 5
5
2,5
M3-Gewinde 8mm tief
100
100
Φ59
Φ86
A
A
100
100
1
A-A (1 : 1)

Lager-Ebene

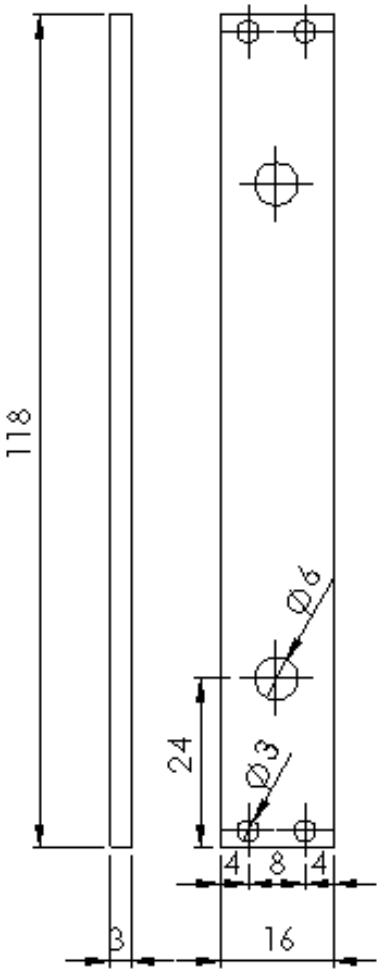

4
9
17,5
18
68
12
64
30
25
98
175
65
12
Phase für
Senkkopfschraube
Ø4
152
10
10
3
3
Grundplatte



Boardhalterung

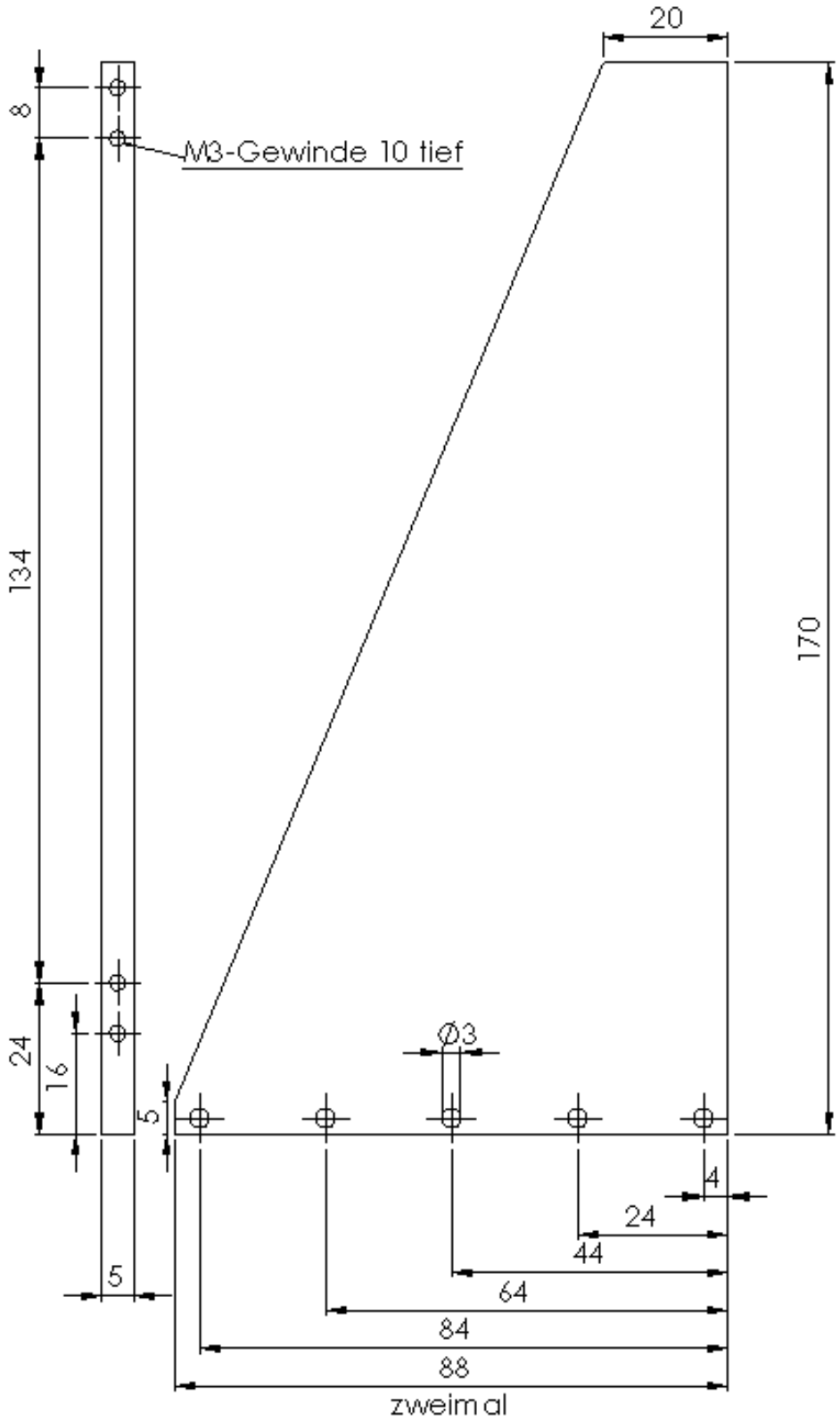


Rotations-Ebene - IBEO



zweimal

Scanner-Halteplatte - IBEO



Seitenstütze - IBEO

4 CD-Inhalt

CD
|–Abbildungen
| – *Abbildungen1-26.zip*
|–Dokumente
| |– Board
| | |– *SBC6000X Hardware Manual.pdf*
| | |– *SBC6000X Linux Application Development Guide v1.2.pdf*
| | – *SBC6000X Linux User Manual v1.2.pdf*
| |– Laserscanner Hokuyo
| | |– *commspec2_eg.pdf*
| | – *URG-04LX_Specifications_Revised.pdf*
| |– Laserscanner – IBEO
| | |– *datasheet_ibeoLUX_de.pdf*
| | |– *Ethernet_protocol_docu.pdf*
| | – *Manual_ibeoLUX_deu.pdf*
| – Schrittmotor
| |– *PD-110-42_ShortSpec.pdf*
| |– *PD-110_kurzanleitung.pdf*
| |– *PD-110_short_guide.pdf*
| |– *PD-110_start.pdf*
| |– *PD-110_start_deutsch.pdf*
| |– *TMCL_reference.pdf*
| – *TMCM-110_manual.pdf*
|– Software
| |– Board
| | |– sam-ba
| | |– *arm-linux-gcc-3.4.5-glibc-2.3.6-linux.tar.bz2*
| | |– *board-sbc9261.c*
| | |– *linux-2.6.24.tar.gz*
| | |– *rootfs_sbc6000x.tar.bz2*
| | |– *sbc6000x_defconfig*
| | – *uImage*
| |– Laserscanner Hokuyo
| | |– URG_Configurer for Win Ver 1.2
| | |– *URG_USB_DRIVER_Linux.zip*
| | |– *URG_USB_DRIVER_Win.zip*
| | |– *vmon0905.zip*
| | – *vmon_linux.zip*
| |– Schrittmotor
| | – TMCL-IDE
| |– *diplom*
| |– *diplom.c*
| |– *menu-k*
| |– *menu-k.c*
| – *profile*

|– SolidWorks
| |– *Baugruppe-Hokuyo.SLDASM*
| |– *Baugruppe-IBEO.SLDASM*
| |– *Boardhalterung.SLDPRT*
| |– *Grundplatte.SLDPRT*
| |– *Kupplung.SLDPRT*
| |– *Lager-Ebene.SLDPRT*
| |– *Motor.SLDPRT*
| |– *Motor-Ebene.SLDPRT*
| |– *Motorsteuerung.SLDASM*
| |– *Nadellager.SLDPRT*
| |– *Roboterplattform.SLDPRT*
| |– *Rotations-Ebende-Hokuyo.SLDPRT*
| |– *Rotations-Ebene-IBEO.SLDPRT*
| |– *Scanner-Halteplatte-Hokuyo.SLDPRT*
| |– *Scanner-Halteplatte-IBEO.SLDPRT*
| |– *Scanner-Hokuyo.SLDPRT*
| |– *Scanner-IBEO.SLDPRT*
| |– *Schleifring.SLDPRT*
| |– *Schleifring-Ebene.SLDPRT*
| |– *Seitenstütze-IBEO.SLDPRT*
| |– *Stütze-Mitte.SLDPRT*
| |– *Stütze-Seite.SLDPRT*
| |– *Welle.SLDPRT*
| |– *Winkel.SLDPRT*
| – *Winkel-Schleifring.SLDPRT*
|– Technische Zeichnungen
| |– *Boardhalterung.png*
| |– *Grundplatte.png*
| |– *LagerEbene.png*
| |– *RotationsEbene.png*
| |– *RotationsEbene-IBEO.png*
| |– *SchleifEbene.png*
| |– *Seitenstütze-IBEO.png*
| |– *Stützen.png*
| – *Winkel.png*
– *Diplomarbeit – Peter Breuer.pdf*