UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

AGAS
Arbeitsgruppe Aktives Sehen

Entwicklung einer Rotationsplattform für den Hokuyo URG-04LX Laserscanner

Studienarbeit
im Studiengang Computervisualistik

vorgelegt von

Christian Delis

Betreuer:	Prof. Dr.-Ing. Dietrich Paulus, Institut für Computervisualistik, Fachbereich Informatik
Erstgutachter:	Prof. Dr.-Ing. Dietrich Paulus, Institut für Computervisualistik, Fachbereich Informatik
Zweitgutachter:	Dipl.-Inf. Johannes Pellenz, Institut für Computervisualistik, Fachbereich Informatik

Koblenz, im Mai 2006

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Die Richtlinien der Arbeitsgruppe für Studien- und Diplomarbeiten habe ich gelesen und anerkannt, insbesondere die Regelung des Nutzungsrechts

Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden. ja ☐ nein ☐

Der Veröffentlichung dieser Arbeit im Internet stimme ich zu. ja ☐ nein ☐

Koblenz, den ..... Unterschrift

Inhaltsverzeichnis

Kapitel 1

Einleitung

Der Hokuyo URG-04LX Laserscanner wird auf der mobilen Roboter Plattform “Robbie” der Arbeitsgruppe Aktives Sehen zur Kartenerstellung und Kollisionsvermeidung eingesetzt. Beim Robocup 2006 konnte das Resko-Team damit sehr gute Karten der Umgebung anfertigen und so die Navigation durch die Arena erleichtern. Die Navigation auf Grundlage der 2D - Scans wird den gewachsenen Anforderungen der Rescue Arenen nicht mehr gerecht, die vielen Rampen führen zu fehlerhaft gemessenen Entfernungen und daraus resultieren ungenaue Karten. Hinzu kommen Hindernisse in unterschiedlichen Höhen, die mit einem statisch montierten 2D - Laserscanner nicht erfassbar sind. Eine Verwendung von kommerziellen 3D - Laserscannern kommt wegen der hohen Anschaffungskosten (> mehrere tausend Euro) nicht in Frage. Idee: Einsatz von mehreren günstigeren 2D - Laserscannern mit unterschiedlichen Blickwinkeln [TMD+06] oder aber die aktive Veränderung der Scanebene [Nüc06]. Das Variieren der Scanebene erfolgt durch Schwenken oder Drehen des Laserscanners [WW03, SD03]. Die Orientierung des Laserscanners im Raum liefert die dritte Dimension.

Im Rahmen dieser Arbeit soll eine Plattform entwickelt werden, die es durch rotative Lagerung des Laserscanners, ermöglicht, 3D-Laserscans der Umgebung zu erzeugen. Hierbei soll ein möglichst einfacher Aufbau erreicht werden, der es weiterhin ermöglicht, den Laserscanner zur Erzeugung von 2D Karten zu benutzen. Um das Stereokamerasystem des Roboters nicht zu beeinträchtigen, wird zusätzlich ein sehr kompakter Aufbau angestrebt.

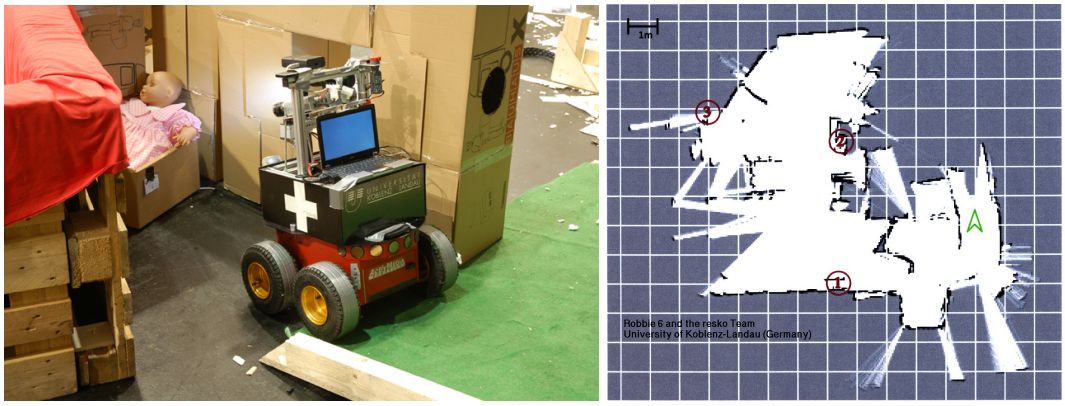


Bild 1.1: *Robbie* beim RoboCup 2006 (links), generierte Karte (rechts)

Der Aufbau der Arbeit ist wie folgt: im Anschluss findet sich eine genauere Beschreibung der vorhanden Arbeiten, in Kapitel 3 folgt die Erklärung über den Aufbau des eigenen Ansatzes. Experimente und eine Berurteilung der Ergebnisse werden in Kapitel 4 aufgeführt, den Schluss bildet Kapitel 5 mit einer Zusammenfassung der Arbeit.

Kapitel 2

Stand der Wissenschaft

2.1 Messverfahren

Um die Entfernung mit Hilfe eines Laserstrahls zu einem reflektierenden Objekt zu bestimmen existieren verschiedene Verfahren:

Time of flight Die meisten Lasermesssysteme beruhen auf dem Prinzip, bei dem die Lauflänge des ausgesendeten Laser Impuls gemessen wird. Aufgrund der hohen Geschwindigkeit des Lichtes (3,3 picosekunden (10^{-12}) für 1 Millimeter) sind dafür sehr genaue Timer notwendig.

Triangulation Bei der Triangulation wird das Objekt durch den Laser angeleuchtet, eine Kamera sucht den Punkt. Der Laser, die Kamera und der Punkt bilden ein Dreieck. Die Entfernung zwischen Kamera und Laser, sowie der Winkel aus der „Laserecke“ sind bekannt. Der Winkel aus der „Kameraecke“ kann durch die Stelle, an dem der Laserpunkt auf die Bildebene projeziert wird, bestimmt werden. Mittels Triangulation wird der Abstand zu dem Objekt berechnet.

Phasenmessverfahren Zur Entfernungsmessung wird die Phasendifferenz herangezogen, diese ergibt sich aus einem Vergleich der Phasenlage des gesendeten Signals zum empfangenen Signal.

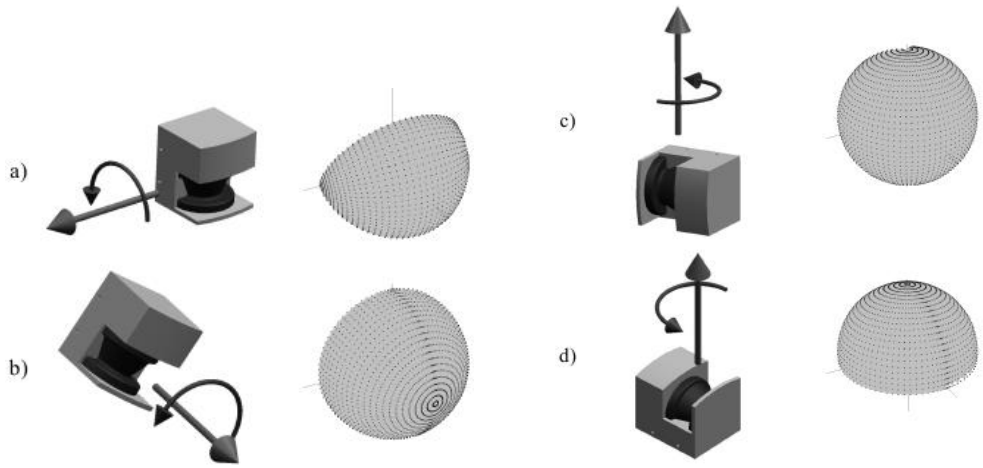


Bild 2.1: Scanmethode (links) und Messwertdichteverteilung (rechts): (a) pitching scan, (b) rolling scan, (c) yawing scan und (d) yawing scan top (*Quelle: [WW03]*)

2.2 Scanmethoden

Die Kombination eines 2D - Laserscanners mit einem Servomotor erlaubt verschiedene Anordnungen der Scanebenen und der Rotationsachsen, die zu unterschiedlichen Messfeldern führen. Abhängig von der Stellung der Rotationsachse zu der optischen Achse des Laserscanners ergeben sich verschiedene Dichtefunktionen der Messpunktverteilung. Die Dichteverteilung ist maximal nahe der Rotationsachse und minimal für Laserstrahlen orthogonal zur Rotationsachse. Wulf et al.(2003) [WW03] vergleicht und benennt vier Kombinationen dieser Scanmethoden:

Während beim *pitching scan* und beim *yawing scan* die Rotationsachse orthogonal zur optischen Achse steht, sind bei dem *rolling scan* und beim *yawing scan top* die Rotationsachse und die optische Achse identisch. Dies bedeutet, dass die höchste Dichte der Messpunkte in der mittleren „Blickrichtung“ des Laserscanners liegt. Die Auswahl der idealen Scanmethode ist somit abhängig von der Anwendung (3D-SLAM, Kollisionsvermeidung), den möglichen Montagestellen und der gewollten Dichteverteilung.

Neben der Unterscheidung nach der Orientierung des Sensorsystems, existieren auch Unterschiede in den Zeitpunkten an denen die Änderung der Scanebene erfolgt. Hier gibt es

Bild 2.2: AIS 3D Laserscanner, Quelle: [SLNH01]

zwei Arten, zu einem die sequenziellen Aufnahmeverfahren die Scannen, Rotieren, Scannen, ... oder die bei der permanenten Rotation Scannen. Während bei den sequenziellen Aufnahmeverfahren die Zuordnung der Orientierung (Rotationswinkel) für jeden Scan erfolgt, so ist es schwierig bei den permanent rotierenden Laserscannern jedem Messpunkt den richtigen Rotationswinkel zuzuordnen. Ein Pluspunkt, der sich ständig rotierenden Systeme, ist, dass hier praktisch keine Totzeiten durch das Zurückdrehen oder die Positionierung entstehen.

Nachfolgend werden zwei Beispiele zu verschiedenen Scanmethoden vorgestellt:

2.3 Pitching Scan

Grundlage des in "Aufbau eines 3D - Laserscanner für autonome mobile Roboter" [SLNH01] beschriebenen 3D-Laserscanners ist ein 2D-Laserscanner der Firma Schmersal, der drehbar gelagert wurde (Abbildung 2.2). Die Rotationsmöglichkeit ist quer zur optischen Hauptachse angebracht, dies ermöglicht eine Drehung senkrecht zur Blickrichtung. Der Laserscanner ist über die serielle Schnittstelle (COM1) mit einem PC verbunden, auf dem ein Real-Time Linux installiert ist. Das Echtzeitbetriebssystem übernimmt die zeitkritische Steuerung des Servomotors, der über die parallele Schnittstelle mit dem PC verbunden ist. Der so erfassbare Bereich umfasst $150^\circ \times 90^\circ$bei einer Winkelgenauigkeit von 0.5° und einer Scangenauigkeit von 5cm, dadurch können Auflösungen von bis zu 113400 Messpunkten erreicht werden.

Weitere Technische Daten des 3D-Laserscanners:

Grösse (B $\times$ H $\times$ T)	$350mm \times 240mm \times 240mm$
Gewicht	$4,5kg$
Betriebsspannung Laserscanner	$24V$
Drehmoment Servomotor	$210Ncm$
Reichweite	$60m$

2.4 Yawing scan top

Der permanent rotierende RoSi-Scanner (RoSi = Rotating Sick) [SD03] der Universität Karlsruhe besteht aus dem LMS 200 2D-Lasermesssystem der Firma Sick, das durch eine 10-Kanal Drehdurchführung mit einer Motor-Getriebe-Kombination gekoppelt ist. Hierbei entspricht die Rotationsachse der optischen Achse des Scanners. Die Signale des optisch inkrementellen Enkoders werden quadraturkodiert an die Auswerteelektronik weitergeben. Zur Kommunikation des Scanners mit dem Standard-Embedded-PC wurde dieser um eine modifizierte RS-422 Interface Karte erweitert. Als Betriebssystem kommt ein normales Linux zum Einsatz.

Bild 2.3: RoSi = Rotating Sick, Quelle: [SD03]

Weitere Technische Daten des LMS200 von Sick:

Grösse (B $\times$ H $\times$ T)	155mm $\times$ 210mm $\times$ 156mm
Gewicht (nur Laserscanner)	4,5 kg
Betriebsspannung	24V
Genauigkeit	$\pm$2cm
Reichweite	80m

Der so erfassbare Bereich umfasst 100° $\times$ 360°, bei einer Auflösung von 0,25° innerhalb der Scanebene, die rotatorische Winkelauflösung beträgt zwischen 0,9° und 3,6°.

Kapitel 3

Aufbau und Realisierung der Rotationsplattform

Der vorhandene Aufbau des Roboters erlaubt die Montage des 3D-Sensorsystems mit der Scanmethode *yawing scan top* oberhalb des Stereokamerasystems oder unterhalb als *pitching scan* oder *rolling scan* (Vergleich Scanmethoden: 2.2). Da der 2D-Laserscanner weiterhin als solcher benutzt werden soll und ein möglichst niedriger Schwerpunkt des Aufbaus angestrebt wird, da der Roboter sonst auf Rampen umkippen und die teuren Sensoren beschädigen könnte, fällt die yawing scan top Montage raus. Desweiteren befindet sich die größte Dichteverteilung bei dem *yawing scan top* an der Decke, was in engen Fluren sehr gut als Orientierung dienen kann, jedoch in den Rescue Arenen mit Deckenhöhen ausserhalb der Reichweite des Laserscanners nicht sinnvoll ist. Bei den verbliebenden Scanmethoden, *pitching scan* und *rolling scan*, blickt der Laserscanner in Fahrtrichtung und kann so weiterhin als 2D-Laserscanner eingesetzt werden. Der *pitching scan* erzeugt zwei Regionen mit einer hohen Messwertdichte, jedoch liegen diese in den weniger interessanten Bereichen links und rechts an der Wand.

Im Gegensatz dazu liefert der *rolling scan* die höchsten Messwertdichten in der mittleren Blickrichtung des Laserscanners bzw. in der Blickrichtung des mobilen Systems und abnehmende Dichten zum Rand hin. Unter der Vorraussetzung, dass der Roboter in die Richtung mit dem größten Interesse schaut, ist die Messpunktverteilung in diesem Sinne

ideal. Ein Vorteil des *pitching scan*, im Vergleich zum *rolling scan*, ist, dass die Messwerte Horizontal und Vertikal angeordnet sind und nicht radial, dies erleichtert die Segmentierung und Objekterkennung. Der *rolling scan* kann durch eine einfache Drehung um 90° auch als *yawing scan top* verwendet werden, was für andere Anwendungen der Arbeitsgruppe sinnvoll sein kann, da solch ein 3D-Scan die komplette obere Hemisphäre abdeckt. Aufgrund der Messwertverteilung und der größeren Flexibilität für spätere Anwendungen wurde sich für die *rolling scan* Methode entschieden.

Unabhängig von der Art der Montage musste geklärt werden, ob ein sich ständig rotierender Laserscanner oder ein sequenzielles Aufnahmeverfahren in Frage kam. Ein permanent rotierender Laserscanner setzt allerdings ein Echtzeitbetriebssystem vorraus, da die Zeit zwischen dem Auslesen des Messwertes und der Übertragung der Position des Laserscanners gering sein bzw. mit einem konstanten Zeitversatz erfolgen soll. Die Bestimmung des Zeitversatzes ist für die genaue Zuordnung zwischen Winkel und Messpunkt wichtig, da der Laserscanner während dieser Zeitspanne weitergedreht wird. Der Zeitpunkt der Messung ist aufgrund der blockweisen Übertragung des Laserscanners und des fehlenden Zeitstempels nicht feststellbar. Auch die Vielzahl der anderen Geräte, die gleichzeitig mit dem USB-Bus des Notebooks kommunizieren, sorgen dafür, dass die Daten eine unbestimmte Zeit auf der Leitung verbleiben, diese Zeit lässt sich durch die unvorhersehbare Kommunikation anderer Geräte auch durch Kalibrierung nicht herausfiltern. Das Notebook der Arbeitsgruppe verwendet kein Echtzeitsystem und die Anbringung eines zusätzlichen PC104-Board oder Mini-ITX-Board kam aus Platzgründen und wegen des zusätzlichen Stromverbrauchs nicht in Frage, daher wurde sich gegen einen sich permanent rotierenden Laserscanner entschieden.

Als nächstes musste ein geeigneter technischer Aufbau gefunden werden um den Laserscanner nach jedem Scan ein Stück weiter zudrehen. Aufgrund des Öffnungswinkels von 240° des Laserscanners reicht eine Drehung um 180° aus, um einen vollständigen 3D-Scan der Umgebung zu erhalten. Nach jedem Scan schickt der Laserscanner innerhalb der Totzeit ein Sync Signal (Vergleich Bild: 3.1 rechts), dies sollte genutzt werden, um zu erkennen, wann ein Scan zu Ende ist und gleichzeitig als Signal dienen, den Laserscanner ein Stück weiterzudrehen. Da die Zeit zwischen Start des Sync-Signals und Start des neuen Scans lediglich 29,1ms beträgt, musste eine Lösung her, die es schafft, innerhalb

der vorgegeben 29,1ms die Drehung zu vollenden. Eine Überschreitung dieser Zeit hätte zur Folge, dass nur jeder zweite Scan brauchbar wäre, und die Scanzeit des Gesamtsystems sich dadurch verdoppeln würde. Die zeitkritische Steuerung der Positionierung der Motoren wird aus dem PC ausgelagert und einem Microkontroller Board übergeben.

Aus der Art des Motors ergaben sich zwei Möglichkeiten:

1. Die Verwendung eines Schrittmotors in Kombination mit einem Planetengetriebe. Eine Untersetzung des Planetengetriebes würde eine genaue Positionierung erlauben. Die aktuelle Position könnte mit Hilfe eines Winkelgebers oder Inkrementalgebers festgestellt werden. Das System könnte sich um 360° drehen, dies würde ein zurückdrehen überflüssig machen, jedoch würde es den Einsatz eines Schleifringes erfordern. Um den Schrittmotor anzusteuern ist ein Motortreiber erforderlich. Bei der Drehung um 360° würde der Sensor die Sicht des Stereokamerasystems beeinträchtigen. Dieser Aufbau ist mechanisch komplexer und von der Grösse und dem Gewicht her ungeeigneter als die zweite Alternative.

2. Ein Modellbauservo hat das Planetengetriebe schon integriert und die Positionansteuerung ist durch die Pulsweitenmodulation exakt möglich, dies macht die Verwendung eines Inkrementalgebers überflüssig. Auf einen Schleifring zur Übertragung der Daten kann ebenfalls verzichtet werden, da zur Anfangsposition zurückgedreht wird. Der Modellbauservo erlaubt einen wesentlich kompakteren Aufbau.

Die Überlegungen zum Aufbau haben zu einem sequenziellen Aufnahmeverfahren geführt, das System wird unterhalb des Stereokamerasystems montiert, die zeitkritische Ansteuerung des Servomotors wird von einem Microkontroller übernommen. Im folgenden werden zuerst die Hardwarekomponenten im einzelnen vorgestellt bevor der Aufbau der Rotationsplattform beschrieben wird.

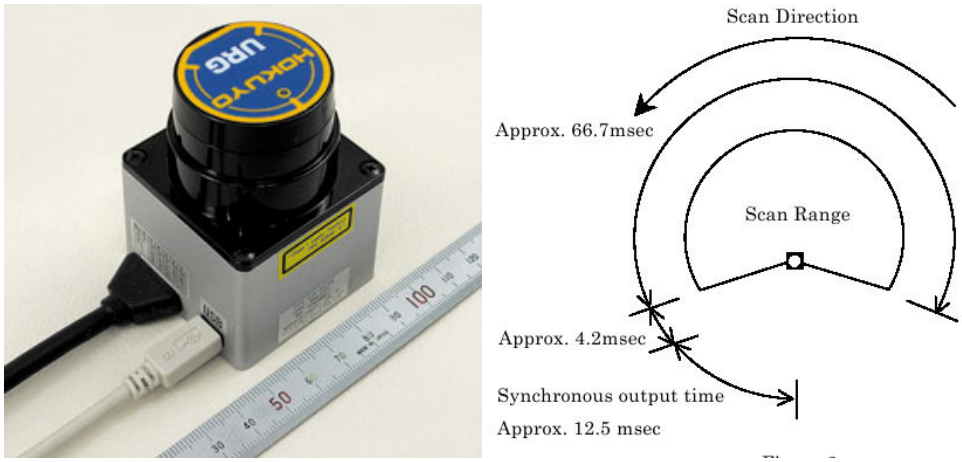


Bild 3.1: Hokuyo URG-04LX (Links), Scanablauf mit Totzeit (Rechts), (Quelle Bild links: [hok06], rechts: [URG05])

3.1 Hardware-Architektur

3.1.1 Hokuyo URG-04LX Laserscanner

Der Hokuyo URG-04LX Laserscanner [hok06] (Abbildung 3.1) ist ein Lasermesssystem für den Innenraum. Als Lichtquelle dient ein Infrarotlaser (λ=785 nm) mit geringem Strahlungsniveau (Laserschutzklasse 1 [1]). Die Entfernungsmessung basiert auf dem weniger durch Oberflächenglanz und Objektfarbe verfälschbaren Prinzip der Phasendifferenzberechnung (Phasenmessverfahren). Der Laserscanner hat eine Reichweite von bis zu 4 Metern[2] bei einer Auflösung von 1mm. Die Genauigkeit beträgt in einer Entfernung von 20 - 1000 mm $\pm$ 10mm und zwischen 1 - 4 Metern $\pm$ 1 %. Bei einem Öffnungswinkel von 240° und einer Scanauflösung von 0.3515625° (360° \1024) besteht eine Scanzeile aus 682 Messpunkten. Die Dauer eines Scans beträgt 100 ms.

Zu den besonderen Merkmalen zählen die geringen Abmessungen von 5 cm $\times$ 5 cm $\times$ 7 cm (B $\times$ T $\times$ H) und sein geringes Gewicht von 160 gr. im Vergleich zu Sick Laser-

[1] DIN EN 60825-1

[2] ab Firmware $>$ Ver3.0.00 bis zu 5,6 Meter

scannern. Die Verbindung mit dem PC kann wahlweise über USB 2.0 oder RS232C erfolgen, wobei die Stromversorgung über den USB Anschluss nicht ausreichend ist. Im USB-Betrieb werden die Daten über USB versendet, die externe Stromversorgung erfolgt durch einen Steckverbinder (Abbildung 3.1), über den auch das Synchronisationsignal (Pinbelegung Tabelle 3.1) abgegriffen werden kann.

Pin	Semantik
1	NC
2	NC
3	Output (Sync Signal)
4	GND
5	RxD
6	TxD
7	0V
8	5VDC

Tabelle 3.1: Pinbelegung des Steckverbinders Quelle: [URG05]

Die Scandauer von 100 ms pro Scan setzt sich aus 66,7 ms echtem Scannen und 32,3 msec Totzeit, in der sich der Spiegel im „blinden“ Bereich befindet, zusammen (Abbildung 3.1 rechts). Dem aktiven Scannen folgt eine Ruhepause von 4.2 ms, danach gibt der Scanner ein Synchronisationssignal mit einer Dauer von 12,5 ms aus.

Die Datenkommunikation wird durch den Host gestartet, das Kommando setzt sich aus dem Kommandobuchstaben gefolgt von Parametern mit einem abschliessenden LF oder CR zusammen. Der Laserscanner sendet das Kommando zurück und hängt den Status und die Daten an das Kommando (Vergleich Tabelle 3.2), die Daten werden in Blöcken zu je 64 Bytes gesendet.

Mit dem *G*-Kommando (Tabelle: 3.2) wird der Scanvorgang angestoßen, für weitere Kommandos siehe [URG04]. Der Startpunkt gibt die „Step“an, von der die Daten bis zu dem Endpunkt übertragen werden. Die Clustergröße gibt an, wieviele benachtbarte Messpunkte der Sensor zu einem Cluster gruppiert, wobei er den kleinsten Wert überträgt. Ist der Status der Anwort ungleich 0, handelt es sich um einen Fehlercode. Die Daten bestehen

Host ⟶ Sensor

G	Start Punkt (3 Ziffern)	Endpunkt (3 Ziffern)	Cluster Anzahl (2 Ziffern)	LF or CR

Sensor ⟶ Host

G	Start Punkt (3 Ziffern)	Endpunkt (3 Ziffern)	Cluster Anzahl (2 Ziffern)	LF or CR
Status		LF		
Datenblock 1 (64 Bytes)		LF		
...		LF		
Datenblock N-1 (64 Bytes)		LF		
Datenblock N (n Bytes)		LF	LF	

Tabelle 3.2: Scan Kommando Quelle: [URG04]

aus der gemessenen Entfernung. Zur Übertragung teilt der Sensor die 12 Bit Binärzahl in zwei 6 bit Zahlen und konvertiert diese in je 1-Byte, dies geschieht, in dem er zu jeder Zahl 0x30 hinzuaddiert. Zum Dekodieren wird 0x30 subrahiert und nach dem Big Endian System die Zahlen zusammengeführt.

Beispiel:
$1234mm = 010011010010_2$
$\downarrow$ Teilung
$(010011_2, 010010_2) = (13H, 12H)$
$\downarrow$ +30H
$(43H, 42H) = (C, D)$

Die Winkelauflösung entspricht 0.3515625°, das heisst 768 Messpunkte für das 270 Grad Modell, bei dem 240 Grad Modell erfolgt die Messung zwischen der 44. und 725. Step. Die Steps[3] 0, 384, 768 entsprechen dabei -135°, 0° und 135°in Bezug zur Frontachse (Abbildung 3.2). Die Umrechnung der Steps s in Grad erfolgt durch:

$$-135° + (s * 360°/1024) \tag{3.1}$$

[3] papers/datasheets/laserscanner/URG-LX_StepAndAngle.pdf

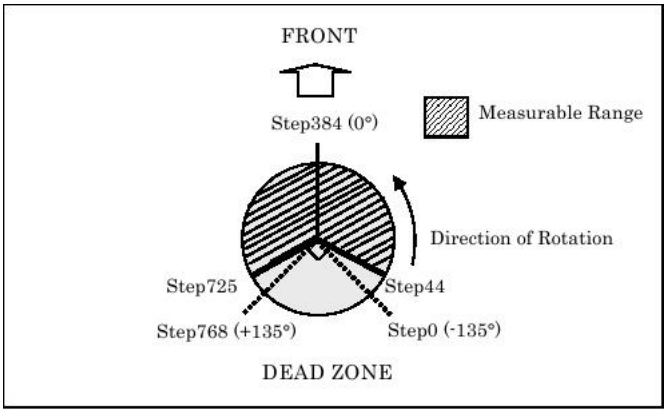


Bild 3.2: Bedeutung der Steps, Quelle: [URG04]

3.1.2 Mikrocontroller Board

Für die Auswahl des Microcontroller Boards waren folgende Anforderungen entscheidend:

- USB Anschluss, da das Notebook auf dem Roboter über keine seriellen Schnittstellen verfügt.
- Auf die Verwendung eines zusätzlichen seriell nach USB Umwandlers sollte wenn möglich verzichtet werden.
- Um die Akkus des Roboters zu schonen, sollte das Board ohne eine externe Stromversorgung auskommen.
- Einfache Programmierung des Microcontrollers.
- Zugriff auf die Standardfunktionen des Microcontrollers, kein großes allzweck Experimentierboard (z. B. mit Motortreiber, I^2C Schnittstelle usw.).
- Kompakte Bauweise, geringes Gewicht.

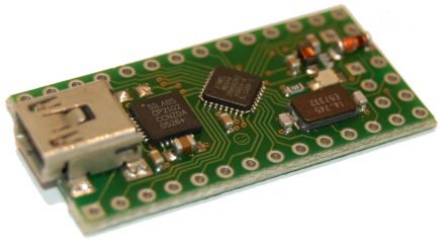


Bild 3.3: crumb8-USB, Quelle: [cru07]

Nach langer Recherche wurde das *Crumb8-USB Rapid Prototyping*[4] Modul gefunden, was den oben genannten Anforderungen entsprach. Das Board ist mit einem Atmega8 AVR Microkontroller von Atmel ausgestattet. Neben einem externen Quarz mit 14,7456 MHz verfügt das Board über eine Status-LED die mit dem PB2 Pin des ATmegas verdrahtet ist. Der UART (Universal Asynchronous Receiver Transmitter) des ATmega8 ist direkt an den *CP2101 USB-TO-UART* Konverter von Silicon Laboratories[5] angeschlossen, so dass das Modul über die Mini-USB Buchse mit dem PC verbunden werden kann. Die Stromversorgung erfolgt über den USB Anschluss.

Alle Signale/Pins des ATmega8 sind nach aussen geführt, mit 18mm $\times$ 36mm entspricht die Größe einem Standard DIL28 Modul. Das Raster von 2,54mm entspricht den üblichen Maßen, so dass das Modul wie ein DIL28 Gehäuse auf eine Platine gelötet werden kann. Desweiteren befinden sich auf dem Board Platzhalter für den ISP Stecker und einen Reset Jumper. Der Reset Jumper ermöglicht einen manuellen Reset des Microkontrollers. Der ISP (In-System Programming) Anschluss erlaubt eine Programmierung des Microcontrollers im eingebauten Zustand.

Der ATmega8 RISC Microkontroller besitzt 8Kbyte nicht flüchtigen Flash Speicher als Programmspeicher, 1Kbyte Arbeitsspeicher (SRAM) und zusätzlich 512byte nicht flüchtigen EEPROM Speicher. Für die 32 je 8-Bit breiten Register stehen 130 interne Befehle[6]

[4] http://www.chip45.com/Crumb8-USB

[5] www.silabs.com

[6] papers/datasheets/microcontroller/AVRBefehle.pdf

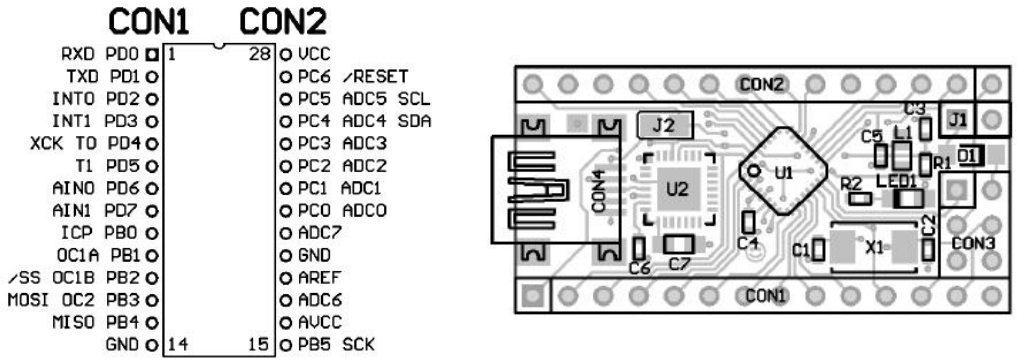


Bild 3.4: crumb8-USB Pins und Schema, Quelle: [cru07]

zur Verfügung.

Der Microkontroller ist programmierbar in Assembler, C/C++, Pascal und BASIC, siehe Anhang B.

Weitere Eigenschaften im Überblick:

- Zwei 8-Bit Timer/Counter
- Ein 16-Bit Timer/Counter
- Ein 10-Kanal ADC
- Zwei externe Interrupt Quellen

Um das Crumb8-USB Modul einsetzen zu können, musste es noch vorbereitet werden. Damit das Modul den Strom über den USB-Bus Anschluss bezieht, wurde die Lötbrücke J2 (Bild 3.4 rechts) mit einem kleinen Stück Draht geschlossen. Danach wurden die 6-polige Stiftleiste zur Programmierung und der Reset Jumper auf die Platzhalter CON3 und J1 gelötet. Schließlich wurde noch ein Fassung angelötet damit das Modul später auf einer Platine befestigt werden konnte.

Im Auslieferungszustand ist der interne 1MHz RC Ozillator des ATmega8 eingestellt, durch Verändern sogenannter Fusebits wird der externe Quarz aktiviert. Für die serielle

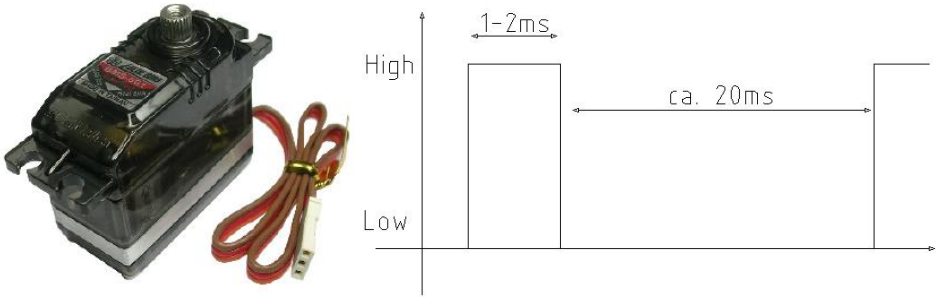


Bild 3.5: Servo und PWM, Quelle: [**?**]

Kommunkation ist das von Bedeutung, da der externe Quarz eine viel höhere Genauigkeit besitzt, zusätzlich ist die Geschwindigkeit des Moduls ca. 15 mal so hoch.

Im Anhang B wird beschrieben wie ein Programm auf den Microkontroller gespielt wird und wie das Setzen der Fusebits geschieht.

3.1.3 Servomotor

Bei der Auswahl des geeigneten Servos waren zwei Dinge wichtig. Zu einem war die Schnelligkeit (Stellzeit) ein entscheidendes Kriterium, da innerhalb der Totzeit der Laserscanner weitergedreht werden muss und bei geringer Anzahl von Scans die Gradzahl, um die gedreht werden muss steigt. Der Modellbauservo sollte so robust sein, den Belastungen auf Dauer standzuhalten. Der Stell- und Haltemoment sollte groß genug sein, den Sensor + Halterung zu drehen und an der Position halten zu können.

Der *BMS-661MG+HS Super Fast*[7] der Firma Bluebird dreht 60° in 0,10sek/0,08sek (4,8V/6V) und verfügt über ein Stellmoment von 50Ncm/62Ncm (4,8V/6V). Die Stromaufnahme beträgt 350mA bei einer normalen Drehung und steigt auf 1150mA um die Position zu halten. Zwei Metallkugellager und das Metallgetriebe machen den Servo robust gegen die mechanischen Belastungen. Ein Kühlkörper schützt den Servo vor Überhitzung.

Die Ansteuerung des Servos erfolgt durch ein pulsweitenmoduliertes Signal. Mit einer

[7] papers/datasheets/servo/BMS-661MG.pdf

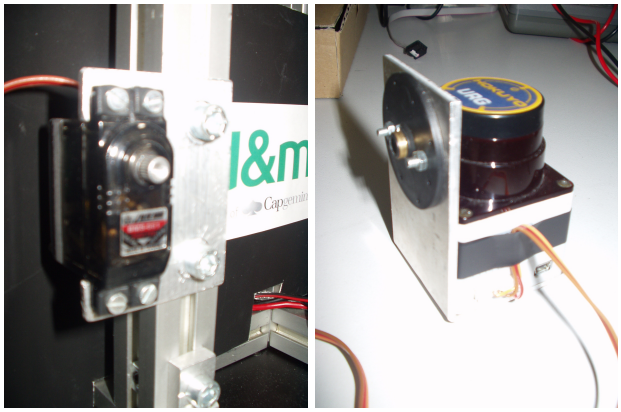


Bild 3.6: Servohalterung und L-Arm

Frequenz von etwa 20ms werden Impulse von 1-2ms Dauer an den Servo geschickt. Ein Impuls von 1 ms Dauer bedeutet linker Anschlag, 2 ms bedeutet rechter Anschlag, die Mittelstellung entspricht so einer Impulsdauer von 1,5 ms. Solange der Servo ein Signal bekommt versucht er diese Position zu halten.

3.2 Aufbau Rotationsplattform

Das nachfolgende Kapitel widmet sich der Anfertigung der Halterungen, anschliessend wird der Zusammenbau sowie die Verkabelung der elektronischen Bauteile näher beschrieben. In Abschnitt 3.3.1 wird auf die Programmierung der einzelnen Komponenten und der Ablauf des Scanvorgangs eingegangen.

3.2.1 Mechanischer Aufbau

Der Aufbau der Rotationsplattform besteht aus zwei Teilen, zum einen der Halterung für den Servo, die an den Aufbau des Roboters montiert wird und zum anderen aus dem Teil, an dem der Laserscanner festgeschraubt wird. Die Servohalterung wurde so konstruiert, dass sie sowohl horizontal wie auch vertikal an den Aluminiumprofilen in der Mitte des Roboters angebracht werden kann. Die Höhe des Gesamtsystems kann dadurch einfach

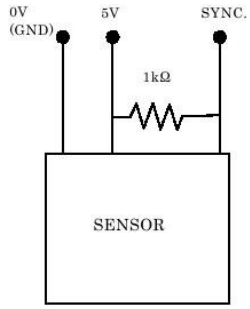


Bild 3.7: Open Collector Ausgang, Quelle: URG_Sync_Signal.pdf

variert werden. Für die Halterung, die den Laserscanner aufnehmen sollte, wurde, da die Gewinde sich an der Bodenplatte des Laserscanners befinden, eine L-Form gewählt. Gemäß den zuvor angefertigten Zeichnungen (Anhang A) wurden aus 3 mm starkem Aluminiumblech die Halterungen ausgesägt, das L gebogen und die Löcher für die Schrauben gebohrt. Abbildung 3.6 zeigt den Aufbau.

3.2.2 Elektronischer Aufbau

Zuerst wurde mit Hilfe eines Ozilloskop festgestellt, wie das Sync-Signal des Laserscanners ausschaut. Der Synchronisationsausgang des Laserscanners ist ein sogenannter Open Collector Output[8], d.h. der Ausgang befindet sich in einem undefinierten Zustand. Um das Sync-Signal abgreifen zu können, musste mit einem $1K\Omega$ Widerstand der Sync-Ausgang auf High-Pegel gezogen werden (Abbildung 3.7 und 3.9). Das Sync-Signal ist somit konstant auf High-Pegel und fällt während den 12,5 ms (Vergleich Bild 3.1) auf Low-Pegel.

Blockschaltbild Das Blockschaltbild(Bild: 3.8) gibt eine schematische Übersicht über die Verbindung der einzelnen Komponenten und deren Kommunikationskanäle. Stromversorgung und Stromquelle fehlen. Der Laserscanner überträgt die Daten über USB an den PC und liefert nach jedem Scan ein Sync-Signal. Der Microkontroller empfängt das Sync-Signal, überträgt die Position an den PC und steuert den Servo an.

[8] papers/datasheets/laserscanner/URG_Sync_Signal.pdf

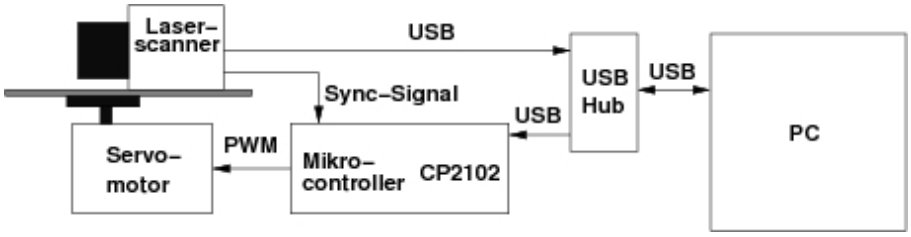


Bild 3.8: Schematische Darstellung der Hardware und Kommunikationskanäle des 3D-Laserscanners ohne Stromversorgung, Quelle: [PDMP06]

Stromversorgung und Platine Die Stromversorgung des Laserscanners und Servos (5V) erfolgt über einen 12V nach 5V Konverter auf dem Roboter. Normalerweise besitzt der Roboter einen 5V Anschluss, der allerdings während dieser Studienarbeit nicht funktionierte. Die Platine (Abbildung 3.9) hat somit 3 Anschlussleitungen (Strom, Servo und Laserscanner).

3.3 Software-Architektur

Die Software besteht aus drei Komponenten:

- Firmware des Microkontrollers für die Ansteuerung des Servos im 2D und 3D-Modus.
- Kommunikationssoftware auf dem PC für die Anforderung von 3D-Scans.
- Graphisches Benutzerinterface für die Kommunikationssoftware und Weiterverarbeitung (Visualisierung und Auswertung) der Scans.

Nachfolgend werden die einzelnen Softwarekomponenten näher vorgestellt.

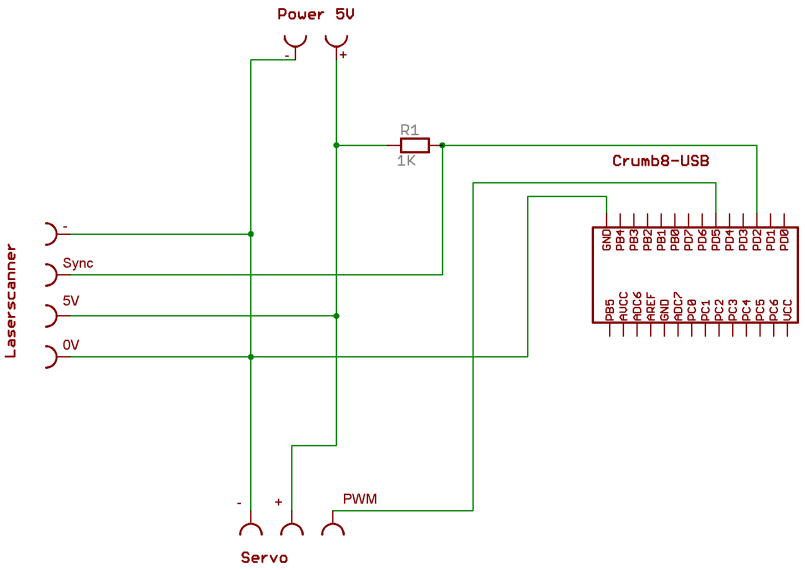


Bild 3.9: Schaltplan Platine

3.3.1 Programmierung des Atmega8-Microkontrollers

Die Programmierung des Atmega8[9] erfolgte in AVR-Assembler[10], um exakter steuern zu können, zu welchem Zeitpunkt was gemacht wird. Mit einer Hochsprache wie C oder C++ ist das nur bedingt möglich. Ein guter Einstieg in die Programmierung mit AVR-Assembler bietet *http://www.mikrocontroller.net/tutorial/*. Das Kompilieren und einspielen der Software auf den Microkontroller, sowie die verwendeten Tools werden im Anhang B beschrieben.

Nach dem Start bzw. einem Reset werden einige Initialisierungen vorgenommen u.a. wird die Baudrate gesetzt und die Empfangs- und Sendeeinheit eingeschaltet, danach wartet der μC auf eine Anfrage. Die Kommunikation zwischen Microkontroller und PC basiert auf dem Client/Server Prinzip. Der Microkontroller fungiert als Server, der auf eine Anfrage des PC's (Client) wartet. Je nach Anfrage geht der Microkontroller in einen von drei Betriebsmodi über:

[9] papers/datasheets/microcontroller/atmega8.pdf
[10] Spezielle Assemblersprache für die Microkontroller von Atmel.

- 3D Scan Steuerung
- Position des Servos wird für den 2D-Betrieb gehalten
- die Softwareversion wird gesendet

Nach der Ausführung wird die Wartestellung wieder eingenommen und auf eine neue Anfrage gewartet.

3D Scan Steuerung

Die Pulsweitenmodulation (PWM) kann durch die eingebauten Timer erfolgen oder mit Hilfe der Software. Während der Programmausführung muss die PWM ständig generiert werden, damit der Servo seine Position hält. Das Programm verwendet die Software PWM. Dies geschieht durch high schalten eines Pins, Dauer des High Signals warten(Anzahl von Takten), Pin Low schalten und 20ms warten. Eine Impulsdauer von 1-2ms reicht nicht aus, um die 180° Drehung des Servos zu erreichen, es musste herausgefunden werden welche Impulsdauer eingesetzt werden muss. Diese wurde mit Hilfe der Programme *lasercontrol_const_start.asm* und *laserscontrol_const_end.asm*, welche den Servo konstant auf der Anfangsposition bzw. Endposition halten, durch Veränderung der Anfangsimpulsdauer und Endimpulsdauer ermittelt. Das Ergebnis der Kalibrierung war, dass die 180° bei einer Anfangsimpulsdauer von 11490 Takten (ca. 0,78ms) und einer Endimpulsdauer von 32140 Takten (ca. 2,18ms) erreicht werden kann.

Nach jedem Scan wird die Impulsdauer des High Signals um die Schrittweite erhöht, so dass der Servo ein Stück weitergedreht wird. Die Schrittweite ergibt sich aus der Differenz zwischen End- und Anfangsimpulsdauer geteilt durch die Anzahl der gewünschten 2D-Scans. Da der Microkontroller nur Ganzzahlen verarbeiten kann entsteht durch die Diskretisierung ein Fehler. Um diesen Fehler auszugleichen, wird der Bresenham Algorithmus benutzt, danach wird zu der Impulsdauer entweder *Schrittweite* oder *Schrittweite+1* hinzuaddiert. Um die Resourcen des Microkontrollers zu schonen wird die Berechnung des Bresenhams auf den PC ausgelagert. Da die Anzahl der Takte größer als Anzahl der 2D-Scans ist, und der Bresenham Algorithmus normalerweise über die größere Zahl

iteriert, konnte dieser nicht direkt verwendet werden.

$$Endimpulsdauer - Anfangsimpulsdauer = Schrittweite * Scananzahl + Rest$$

Als Parameter für den Bresenham Algorithmus wird der Rest und die Scananzahl genommen, der Rest ist immer kleiner als die Scananzahl. Die Schrittweite wird auf den Microkontroller übertragen sowie für jeden Scan der Offset 0 oder 1 als Schrittkorrektur.

Durch Empfang von `REQUEST_3D_SCAN_CTRL` wird die Steuerung für einen 3D-Scan gestartet (siehe auch Abbildung 3.10 Unterzustandsautomat SM 3D_SCAN_CTRL). Der μC empfängt nun die Anfangsimpulsdauer (2 Bytes) als *Low*- und *High*-Byte, diese bestimmt an welcher Position der Servo startet. Die Schrittweite (2 Bytes) folgt darauf bevor die Scananzahl (Anzahl der 2D-Scans, 1 Byte) empfangen und zur Kontrolle zurück gesendet wird. Anschließend wird die Schrittkorrektur empfangen, im Arbeitsspeicher abgelegt und zurück gesendet. Danach wird zur Kontrolle die Anzahl der empfangenen Schrittkorrekturen an den PC zurückgesendet. Nach dem alle erforderlichen Daten vorhanden sind, wird der Servo an die Anfangsposition bewegt. Dies geschieht durch Ausgabe der PWM an Pin *PD5*. Damit der PC erst ein 2D-Scan vom Laserscanner anfordert, wenn der Servo an der Anfangsposition steht, muss das PC-Programm solange angehalten werden. Dies wird durch blockierendes Lesen des Sync-Bytes im PC-Programm erreicht, welches der μC sendet nach dem der Servo an der Anfangsposition steht. Der externe Interrupt wird so konfiguriert, dass die Interruptroutine bei einer fallenden Flanke ausgeführt wird. Der Laserscanner sendet sobald er Strom hat ständig das Synchronisationssignal, daher wird der Externe Interrupt erst jetzt eingeschaltet.

Die Befehle:

- Globalen Interrupt ausschalten,
- Pin PD5 High setzen,
- High Impulsdauer warten,
- Pin PD5 Low setzen,
- Globalen Interrupt einschalten,

- 20 ms warten

werden solange ausgeführt, bis die Scananzahl erreicht ist. Der Microkontroller besitzt einen globalen Interrupt Schalter, mit dem sich alle Interrupts ausschalten lassen, sowie einen Schalter für jeden einzelnen Interrupt. Während der High-Impulsdauer darf keine Interruptroutine ausgeführt werden, da die Dauer eingehalten werden muss. Der Globale Interrupt wird hier ausgeschaltet und nicht der externe Interrupt, weil so durch das Setzen eines Bites in einem Register gespeichert wird wenn während der High Impulsdauer der externe Interrupt auftritt. Sobald der Globale Interrupt wieder eingeschaltet wird, beginnt die Abarbeitung der Interruptroutine. In der Interruptroutine wird die Position (aktuelle Scanzahl) als Byte an den Client gesendet. Zu der aktuellen High-Impulsdauer wird die Schrittkorrektur und die Schrittweite hinzuaddiert, der Servo dreht sich bei dem nächsten High-Impuls dann ein Stück weiter. Bevor die Interruptroutine verlassen wird, wird die aktuelle Position inkrementiert. Wenn die maximale Position erreicht ist, übermittelt der μC die High-Impulsdauer (2 Bytes) zur Kontrolle an den PC. Zum Schluss geht das Programm wieder in die Wartestellung.

2D-Betrieb - Position halten

Sendet der Client `REQUEST_HOLD_2D_POS`, geht der Microkontroller in den Betriebsmodus die 2D-Position zu halten, dazu übermittelt der Client die Halteposition in Anzahl von Takten, die der Highimpuls dauern soll (siehe auch Abbildung 3.10 Unterzustandsautomat SM HOLD_2D_POS). Zur Kontrolle wird das *Low*-Byte zurückgesendet und anschliessend der *UART_RX_Complete* Interrupt eingeschaltet, dessen Interruptroutine nach dem Empfang eines Zeichens ausgeführt wird. Die PWM wird nun generiert, in dem die Befehle:

- Globalen Interrupt ausschalten,
- Pin PD5 High setzen,
- High Impulsdauer warten,
- Pin PD5 Low setzen,

- Globalen Interrupt einschalten,
- 20ms warten

ausgeführt werden bis der Client einen `REQUEST_STOP_2D` sendet. Durch den Empfang wird die Generierung der PWM unterbrochen und in die Wartestellung zurückgekehrt.

Softwareversion senden

Mit `REQUEST_VERSION` kann der Client die aktuelle Softwareversion anfordern, geantwortet wird darauf mit dem String *Lasercontrol Program: Version 1.0*, gefolgt von einem *LF* und *CR* Zeichen. Danach kehrt der Microkontroller wieder in die Wartestellung zurück.

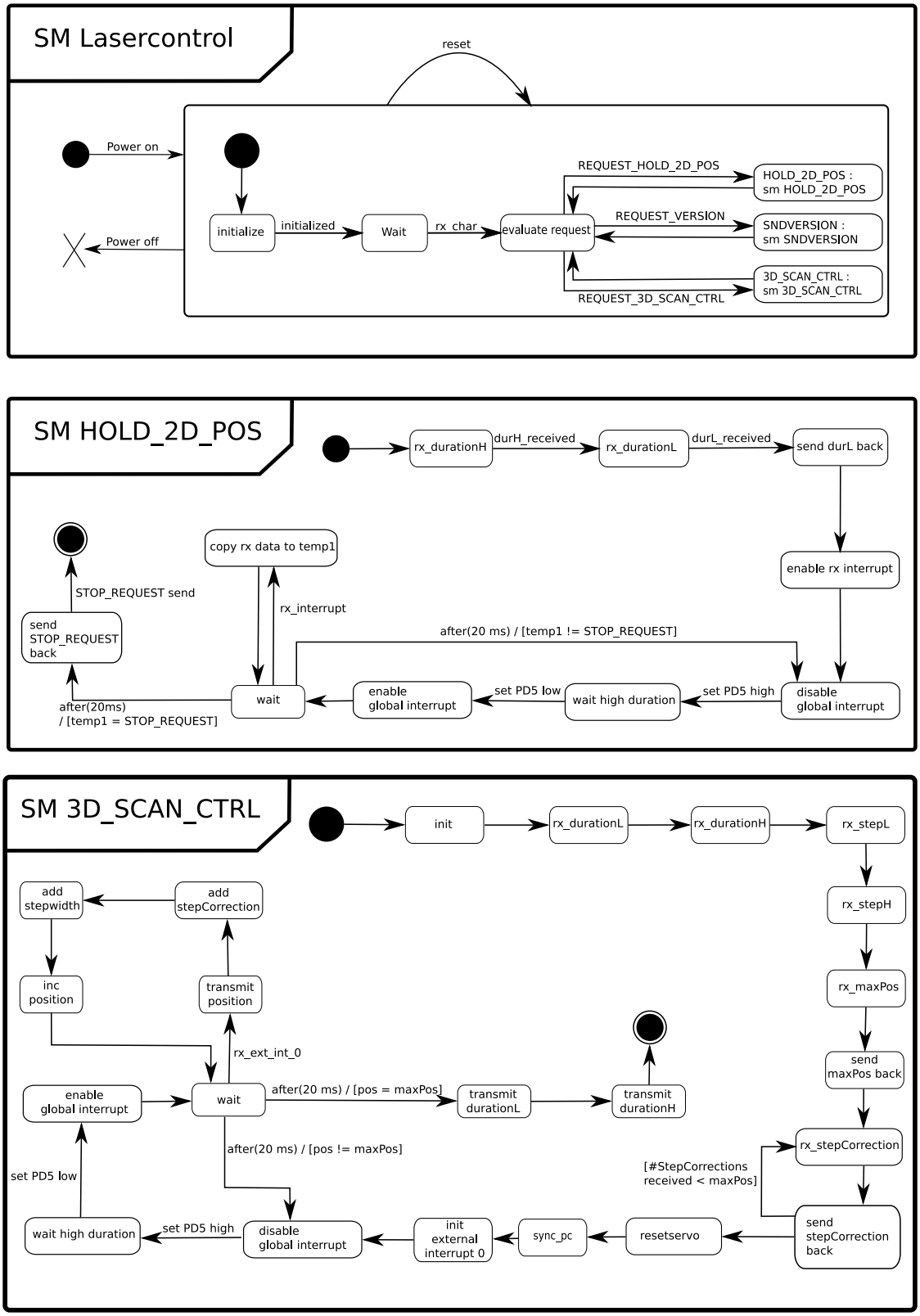


Bild 3.10: Zustandsübergangsdiagramm des Microkontrollers

3.3.2 Programmierung der Kommunikationssoftware

Die Kommunikationssoftware besteht aus der Klasse *DataReader*, die sowohl die Kommunikation mit dem Laserscanner und Microkontroller übernimmt, als auch das Abspeichern und Öffnen von 3D-Scans kapselt. Sie bindet die Datei `config.h` ein in der einige Konstanten definiert sind. Die Klasse benutzt die Klasse Laserscanner3D von Stephan Wirth, welche eine Implementierung des l3d Formats[11] von Stephan Wirth ist. Die 3D-Scans können in dem l3d Format abgespeichert und wieder geöffnet werden.

Der Laserscanner und das Microkontroller Modul können als normale Geräte angesprochen werden (Anhang B). In `void connectDevices()` werden die Geräte geöffnet und mit `bool setupSerial(int aFd, speed_t speed)` die Einstellungen der Flusskontrolle, Baudrate, Frameformat etc. vorgenommen.

Um einen 3D-Scan anzufordern wird die Funktion `void getScanFromLaser(int resolution,int startpos, int endPos, int* numPoints, float** xCoords, float** yCoords, float** zCoords)` verwendet. Als Parameter erhält sie die gewünschte Scananzahl (resolution), die Anfangsimpulsdauer und Endimpulsdauer in Takten. In der Variable numPoints ist nach der Ausführung die Anzahl der gemessenen Punkte und in den Arrays die kartesischen Koordinaten (XYZ) zu jedem Punkt gespeichert. Innerhalb der Funktion werden die Parameter auf Gültigkeit überprüft, die Werte berechnet und nachdem dieser verbunden ist, an den μC übertragen. Die Schrittkorrektur wird berechnet und ebenfalls übertragen. Danach wird das Programm solange mit Hilfe eines blockenden Lesevorgangs angehalten bis der Microkontroller ein Synchronisationsbyte sendet, um anzuzeigen, dass der Servo an der Anfangsposition ist. Nun wird ein 2D-Scan mit `int * urgGetScan(0, 768, 1)` angefordert, die Position als Ganzzahl ausgelesen bis die Scananzahl erreicht ist. Mit Hilfe der Rohdaten wird ein Laserscan3D Objekt `m_myScan` erstellt und die Parameter mit `m_myScan->get3DPoints(numPoints, xCoords,yCoords, zCoords)` durchgereicht.

Für den 2D-Betrieb wird die Funktion `void holdPos(int holdpos)` mit der Position in Anzahl von Takten aufgerufen. Schließlich kann der 2D-Betrieb durch Aufruf von `void stopHoldPos()` unterbrochen werden. Dort wird `REQUEST_STOP_2D` zu

[11] siehe papers/datasheets/l3d_specification/Spezifikation-l3d-1.0.txt

dem Microkontroller geschickt, was ihn veranlasst in die Wartestellung zurückzukehren.

Zum Speichern des aktuellen 3D-Scans wird der Dateiname an die Funktion `void saveScanToL3DFile (QString filename)` übergeben. Das Öffnen einer Datei im l3d Format geschieht mit der Methode `void getScanFromFile(QString filename, int* numPoints , float **xCoords, float** yCoords, float** zCoords)`. Die Datei *filename* wird geöffnet und die kartesischen Koordinaten in den übergebenen Arrays gespeichert.

Umrechnung der Koordinaten

Die Umrechnung der Step s des Laserscanners in den Winkel ϕ (Rad) erfolgt durch:

$$\phi = (-135^\circ + (s * 360^\circ/1024)) * \pi/180$$

Aus der Position p, die der Microkontroller liefert, und der Scananzahl n läßt sich der Winkel θ (Rad) des Laserscanners folgendermaßen berechnen $\theta = p * 180/n$. Für die *Yawing scan top* Methode lassen sich die kartesischen Koordinaten nun einfach aus den Kugelkoordinaten berechnen. Der Radius r entspricht hier der gemessenen Entfernung.

$$x = r * \sin\phi * \cos\theta \tag{3.2}$$

$$y = r * \sin\phi * \sin\theta \tag{3.3}$$

$$z = r * \cos\phi \tag{3.4}$$

Durch Vertauschen von x durch z und z durch $-x$ bekommt man die Umrechnung für die *Rolling scan* Methode.

3.3.3 Programmierung der GUI

Die GUI (*main3DLaserScanner*) dient als Grafische Oberfläche zur Steuerung mit Hilfe der bereits vorgestellten Kommunikationsoftware, sowie der Anzeige und Visualisierung der 3D-Scans. Die Oberfläche wurde in *C++* unter Verwendung von *QT*[12] und *OpenGL*[13]

[12] http://www.trolltech.com/

[13] http://www.opengl.org

Datei	Inhalt
mainwindow.h	Headerdatei der MainWindow-Klasse
mainwindow.cpp	Implementierung der Oberfläche
glwidget.h	Headerdatei der GLWidget Klasse
glwidget.cpp	Implementierung des Fensters zur Visualisierung der Daten
datareader.h	Headerdatei der DataReader-Klasse
datareader.cpp	Implementierung des Kommunikationsprogramm
main.cpp	Hauptprogramm, enthält die main Routine
Laserscan3D.h	Headerdatei der LaserScan3D-Klasse
Laserscan3D.cpp	Implementierung des l3d Formats von Stephan Wirth
config.h	Konstantendefinition
main3DLaserScanner.pro	pro Datei, mit qmake lässt sich das Makefile erzeugen

Tabelle 3.3: Dateien zur main3DLaserScanner GUI

entwickelt (siehe Anhang B). Das Hauptfenster ist ein Objekt der Klasse GLWidget die von QGLWidget abgeleitet ist. Das QGLWidget kann OpenGL Grafiken innerhalb einer QT Applikation darstellen und kann wie ein normales QWidget verwendet werden. In der Unterklasse werden die paintGL(), resizeGL() und initializeGL() Methoden überschrieben. Die GUI befindet sich im Verzeichnis *main3DLaserScanner*, die Dateien sind in Tabelle 3.3 aufgelistet. Die Klasse MainWindow bildet das Gerüst für das GLWidget Objekt. Das GLWidget Objekt besitzt eine Membervariable von Typ DataReader. Die *Signale* von dem MainWindow sind mit den *Slots* des GLWidget verbunden. Die 3D-Scans werden als Punktwolke mit Hilfe der OpenGL Datenstruktur `GL_POINTS` dargestellt. Zur schnelleren Darstellung werden sogenannte Displaylisten verwendet. Die Bedienung erfolgt über das Menü oder das QDockWidget (Abbildung 3.11), das über Menü *View->Control* ein- und ausgeblendet werden kann. Über das Menü *File* lässt sich ein neuer 3D-Scan anfordern, ein vorhandener öffnen oder der aktuelle abspeichern. Die Auswahl der Datei erfolgt über einen Datei Dialog. In dem Menü *View* sind Eigenschaften, die die Anzeige betreffen, zusammengefasst. Die Kantenlänge des Gitters (Grid) entspricht 50 cm in der Wirklichkeit. Die Farben der jeweils positiven Achsen (axis) stehen für weiß = x-Achse, gelb = y-Achse und blau für die z-Achse. Die Verwendung von Displaylisten kann im Me-

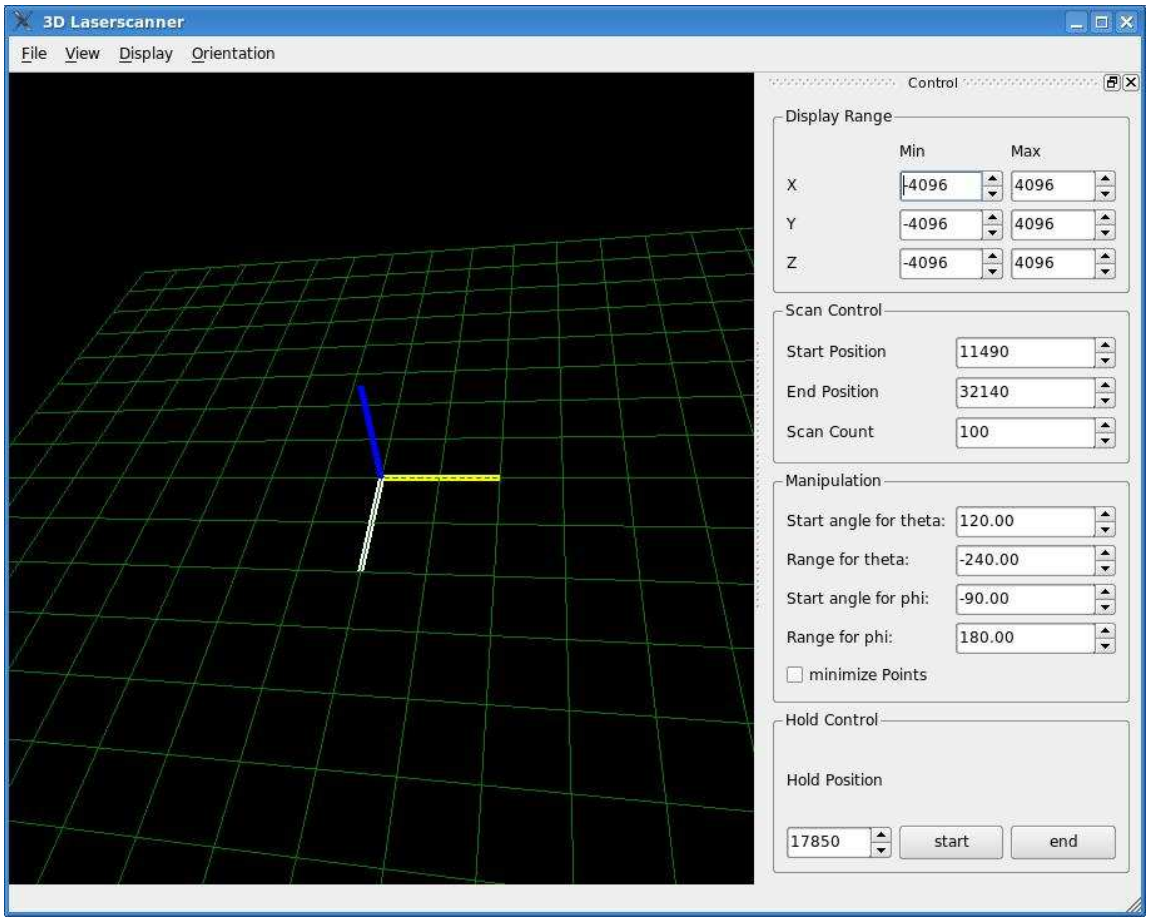


Bild 3.11: main3DLaserScanner GUI

nü *Display* bei Bedarf abgeschaltet werden. Um die Framerate anzeigen zu lassen, muss das Programm zuvor mit der Präprozessor-Direktive *__PTEST__* kompiliert werden. Unter Orientation kann die Anzeige zwischen *rolling scan* und *yawing scan top* umgeschaltet werden.

Über das Dockwidget kann der Bereich eingestellt werden, der angezeigt wird, dabei werden alle Punkte angezeigt, die zwischen dem Min und Max Wert liegen. Um diese Funktion zu nutzen, muss die Verwendung von Displaylisten abgeschaltet werden. Darunter befindet sich die Steuerung für den 3D-Scan, hier kann die Scananzahl verstellt werden, Werte zwischen 40 und 255 sind möglich. Die Manipulation bezieht sich auf die Interpre-

tation der 3D-Scans. Sie ist aus dem l3d Viewer von Stephan Wirth übernommen. Hier lässt sich der Start von Polar- und Azimuthwinkel, sowie der Winkelbereich verändern. Ganz unten befindet sich die Steuerung für den 2D-Betrieb des Laserscanners, hier kann man nur die Position angeben, an der der Laserscanner gehalten wird. Es lassen sich keine 2D-Scans anzeigen. Mit der Maus kann man die Szene rotieren, wenn die Maus bei gedrückter Maustaste über das GlWidget bewegt wird. Durch Drehen des Mausrades kann man in die Szene rein- und rauszoomen.

3.3.4 Kommandozeilenversion der Kommunikationssoftware

Bevor das oben vorgestellte Programm entwickelt wurde, entstand eine Version die in C geschrieben war und die Rohdaten in eine Textdatei schrieb.

Das Programm *laser3dFileOutput.c* bekommt als Parameter die Gerätedatei des Microkontrollers und den Dateinamen, in der die Daten geschrieben werden sollen, übergeben. Die Ausgabe erfolgt in dem Format `Stepnummer Entfernung Position` gefolgt von einem Zeilenumbruch. Die Scananzahl ist innerhalb des Programms über eine Präprozessor Anweisung auf 100 festgelegt, diese ist jedoch veränderbar.

Der Vorteil gegenüber der binären Abspeicherung ist, dass es so viel einfacher ist mit Hilfe von z. B. Perl-Skripten die Daten auszuwerten. Um die Rohdaten in kartesische Koordinaten umzurechnen, wurde ein Perl-Skript (laserDat2Pointfile.pl) geschrieben, das als Argument die Datei mit den Rohdaten und eine Datei für die Ausgabe übergeben bekommt. Innerhalb des Skriptes muss die Variable $DEG_PER_SCAN an die Gradzahl pro Scan angepasst werden. Das Perl-Skript schreibt `x y z` pro Messpunkt in eine Zeile der Ausgabedatei. Die so enstanden Dateien können mit *gnuplot*[14] angeschaut werden.

[14] Plot Programm für Linux

Kapitel 4

Experimente und Ergebnisse

Der Scanner wurde bereits auf dem Robocup eingesetzt. Bisher konnten an der Mechanik keine Veränderungen und an dem 2D-Laserscanner keine Fehlfunktion, bedingt durch die Belastung bei der Rotation, festgestellt werden. Innerhalb kurzer Zeit sind so Aufnahmen des Raumes möglich bei der Objekte in einer Entfernung von bis zu 4 Metern erkannt werden. Die Auflösung ist durch Varieren der Scananzahl zwischen 40 und 255 beliebig einstellbar. Dabei besteht die Punktwolke aus 27.280 bis 173.910 Messpunkten. Tabelle 4.1 zeigt Parameter verschiedener Scanauflösungen sowie die Aufnahmedauer.

Scananzahl	Auflösung (ϕ) (einstellbar)	Auflösung (θ) (fest)	Anz. Punkte	Zeit
40	4,5°	0,352°	27.280	4 sek
50	3,6°	0,352°	34.100	5 sek
100	1,8°	0,352°	68.200	10 sek
200	0,9°	0,352°	136.400	20 sek
255	0,706°	0,352°	173.910	25,5 sek

Tabelle 4.1: Parameter für verschiedene Scan-Auflösungen

Listing 4.1: autoscan.sh

```
#!/bin/bash
#
scancount=100;
sleep 8; # 8 sec to go away
for ((i=0; i<scancount; i++));
do
./laserrawoutput /dev/ttyUSB0 foyer3/foyer3_raw_$i.dat;
sleep 1;
done
```

4.1 Versuchsaufbau

Für die Experimente wurde der 3D-Laserscanner auf eine Tischplatte montiert, dies entspricht der *yawing scan top* Methode. Da hier die höchste Dichteverteilung an der im Labor glatten Decke liegt, konnte so die Scanqualität sehr gut beobachtet werden. Für die einzelnen Scans wurde das grafische Benutzerinterface eingesetzt, da dieses eine bessere Betrachtung der 3D-Scans erlaubt. Um selber nicht auf den 3D-Scans zu erscheinen, wurde die Zeit auf etwa 5,5 sek verlängert, die der Servo in der Anfangsposition gehalten wird. Der Strom für das System wurde über den 12V nach 5V Konverter von einem 12V Bleigel Akku bezogen.

Um mehrere automatisierte Scans der Umgebung zu bekommen, wurde ein Bash Skript (Listing 4.1) geschrieben, dass das Komandozeilenprogramm wiederholt aufruft.

4.2 Ergebnisse

Die am Anfang als problematisch empfundene Unkenntnis über den genauen Austrittspunkt des Laserstrahls erwies sich als nicht so tragisch. Es wurde richtigerweise angenommen, dass der Strahl genau in der Mitte des optischen Fensters austritt. Ein Problem ergibt sich aber, wenn der Laserscanner ein wenig verdreht zu seiner Längsachse montiert ist, was auf Ungenauigkeiten an der Halterung zurückzuführen ist. Deutlich wird das besonders an der Decke beim ersten und letzten Scan (Abbildung 4.1).

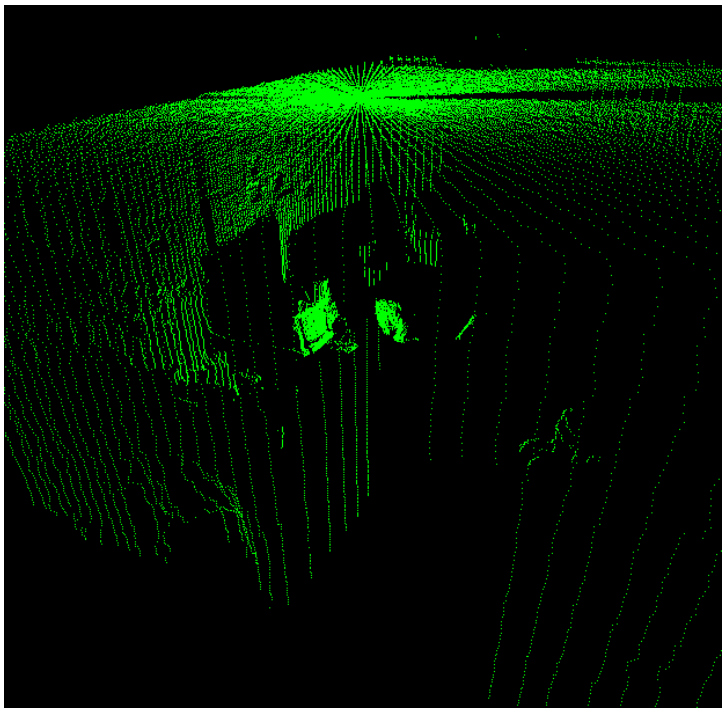


Bild 4.1: fehlerhafte Ausrichtung

Mit dem oben vorgestellten Bash Skript wurden im Foyer des 3. Stockes B-Gebäude 100 Scans hintereinander aufgenommen. Von den 100 Scans wurde zu jedem Messpunkt die durchschnittliche Enfernung berechnet. Der daraus enstandene entrauschte 3D-Scan wurde untersucht und ein Bias im 2D Laserscanner festgestellt (Abbildung 4.2; Vergleich [PDMP06]).

Das Rauschen des Laserscanners lässt sich sehr gut erkennen, wenn die Messpunkte an der Decke in der Blickrichtung der optischen Achse zum Zentrum betrachtet werden. Idealerweise haben die Messpunkte innerhalb eines Ringes untereinander und zur Mitte den gleichen Abstand (Abbildung 4.3).

In dem nachfolgendem Experiment wurde von dem Serverraum Scans in verschiedenen Auflösungen erzeugt. Sie sollten einen Überblick darüber geben, welcher Detailgrad mit verschiedenen Auflösungen erreicht wird (Abbildungen 4.4, 4.5 und 4.6).

Zum Schluss noch eine Aufnahme, die mit am Roboter montiertem Laserscanner enstanden ist (Abbildung 4.7).

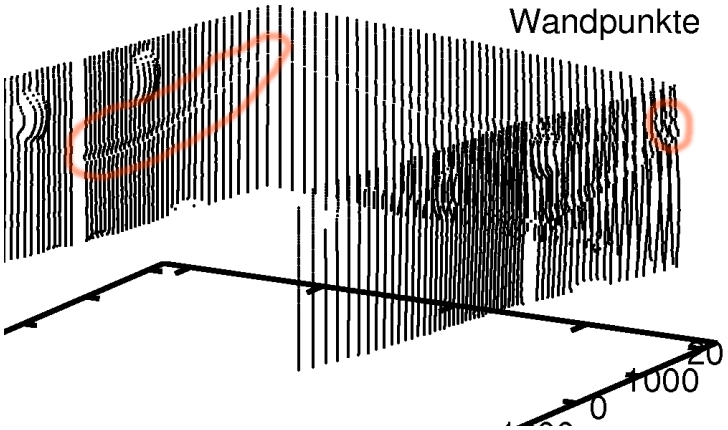


Bild 4.2: Bias Laserscanner, Quelle [PDMP06]

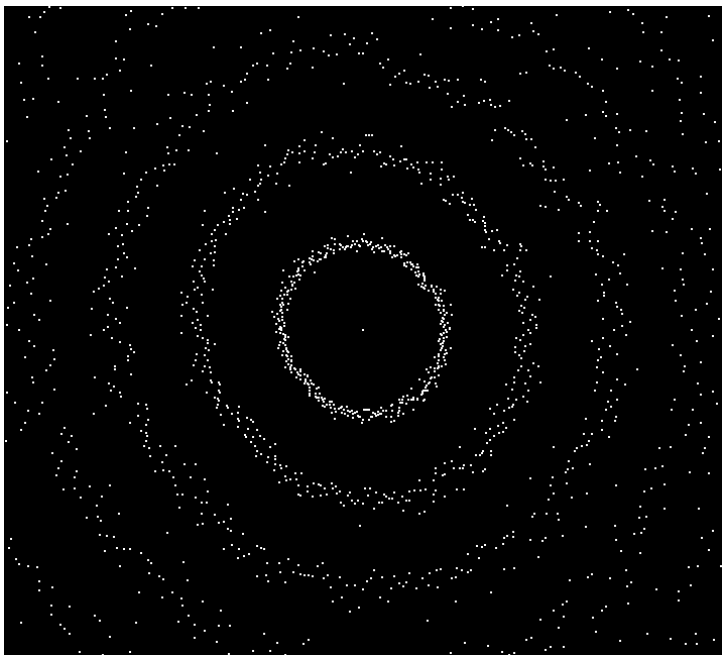


Bild 4.3: radiale Anordnung der Messpunkte

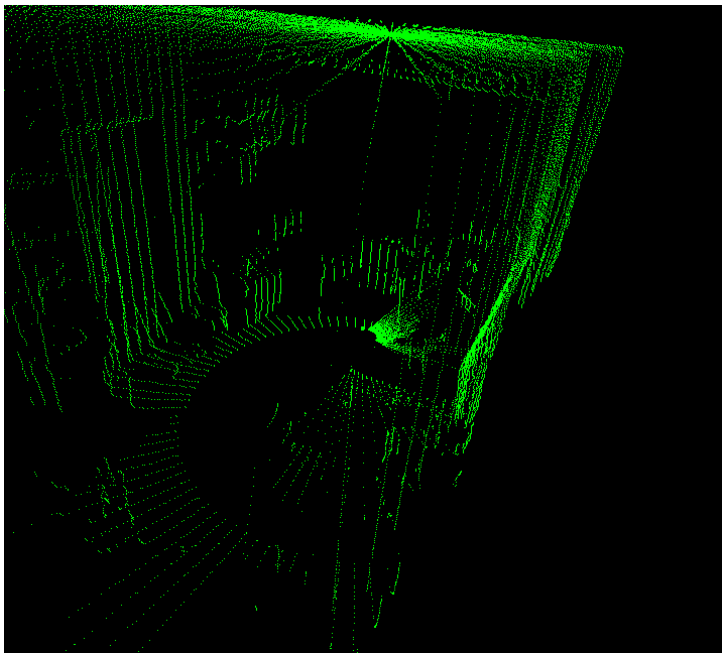


Bild 4.4: Serverraum 34100 Punkte

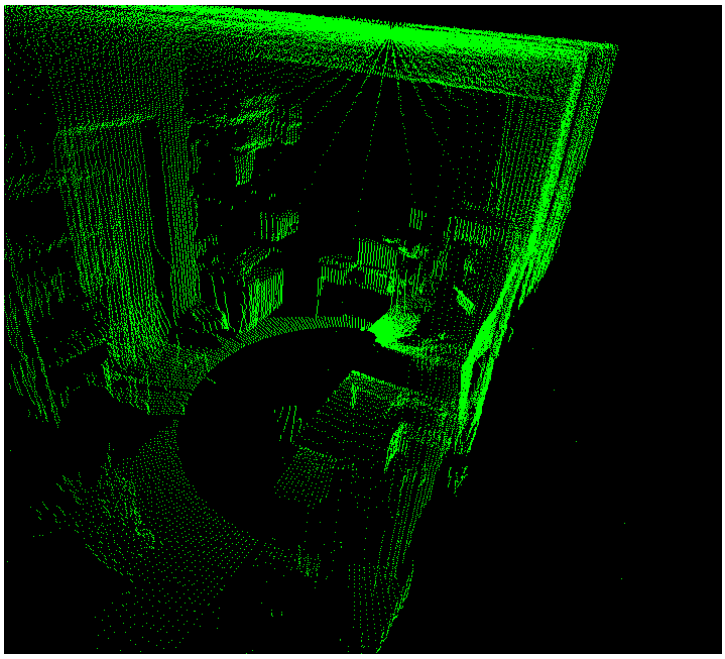


Bild 4.5: Serverraum 102.300 Punkte

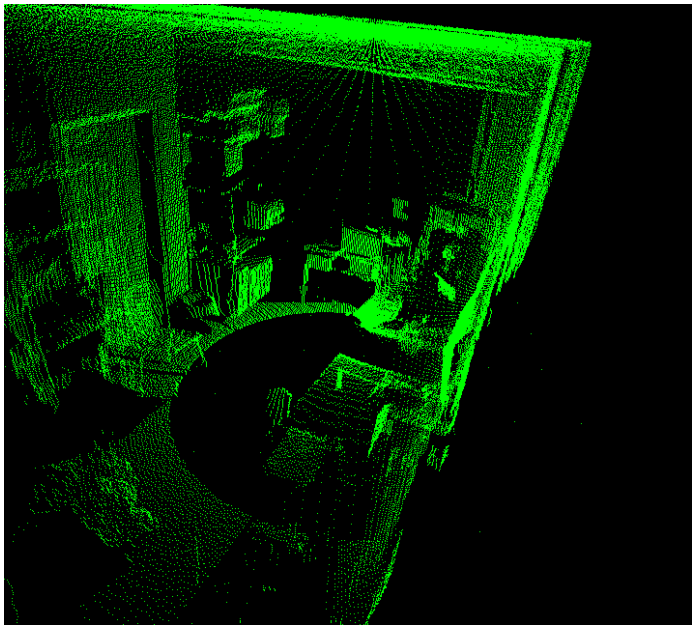


Bild 4.6: Serverraum 173910 Punkte

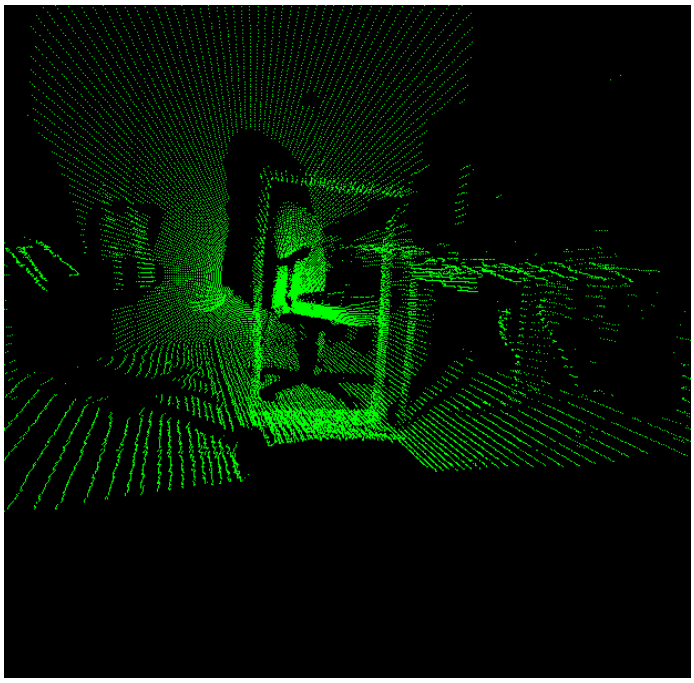


Bild 4.7: Computerlabor Rolling Scan

Kapitel 5

Zusammenfassung

Mit dem hier vorgestellten 3D-Laserscanner auf Basis des Hokuyo URG-LX04 verfügt die Arbeitsgruppe Aktives Sehen der Universität Koblenz über ein Gerät das vielseitig einsetzbar ist. Es ist ein leichter und kompakter 3D-Laserscanner aus Bauteilen für unter 100 Euro enstanden, der trotzdem über eine ausreichende Genauigkeit für die Anwendungen der AGAS verfügt. Der Aufbau wurde so konstruiert, dass das Stereokamerasystem des Roboters nicht beeinträchtigt wird. Die Konstruktion erlaubt es den Laserscanner sowohl vertikal als auch horizontal zu verschieben. Der Laserscanner kann sowohl mit der *Rolling Scan* Methode wie auch mit der *yawing scan* Methode betrieben werden. Im Betrieb der *yawing scan* Methode kann mit einem einzigen Scan die komplette obere Hemisphäre abgedeckt werden. Die Betriebspannung von 5V ermöglicht den Betrieb auf vielen Mobilen Systemen. Die variable Scanauflösung sorgt dafür, dass von Kollisionsvermeidung bis hin zur Modellerstellung alles möglich ist.

Die gelieferten Daten können als Basis für weitere Arbeiten dienen. Die Arbeit an der Fusion von mehreren Scans zu einem großen ist bereits abgeschlossen. Als nächstes ist das automatische Generieren von Texturen mit Hilfe von Kameras und Mappen auf die 3D Daten geplant. Mit der Arbeit an einer aktiven Ansteuerung wurde bereits begonnen. Im Ausblick steht ebenfalls das Tracking von 3D Punkten in Bildern.

Tabelle 5.1 listet die Technischen Daten des AGAS 3D-Laserscanners nochmal zusammenfassend auf und Abbildung 5.1 zeigt den Roboter mit montiertem Laserscanner.

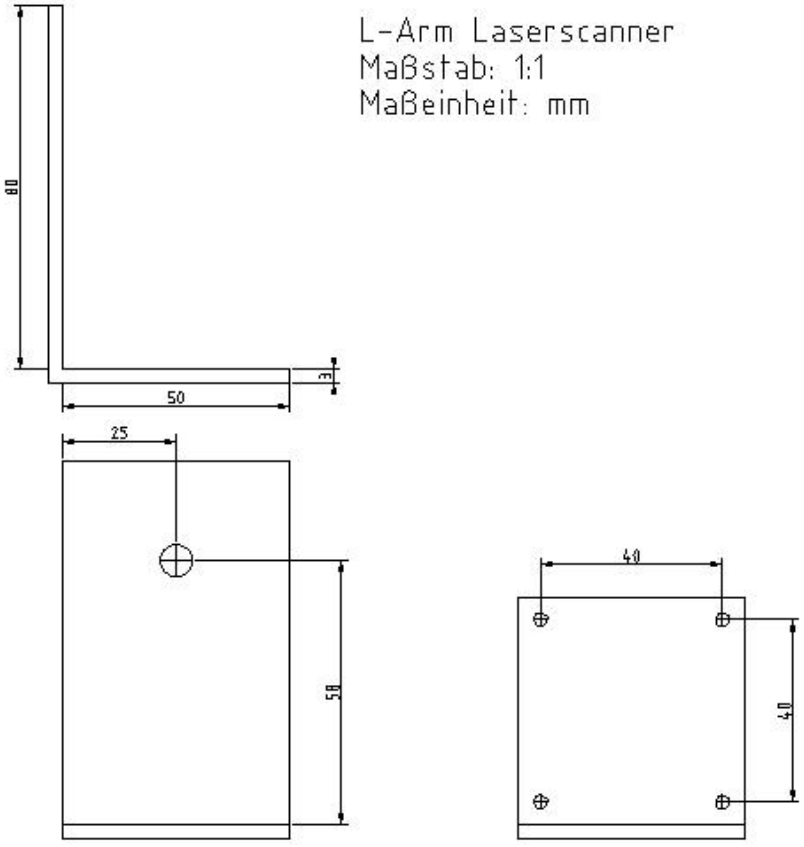
Gewicht:	ca. 500 gr.
scanbarer Bereich:	240° × 360°
Winkelauflösung:	0.352°
Scangenauigkeit:	±1%
Auflösung:	27280 bis 173910 Punkte
Scandauer:	4-25,5 s
Reichweite:	4 m

Tabelle 5.1: Technischen Daten des AGAS 3D-Laserscanners

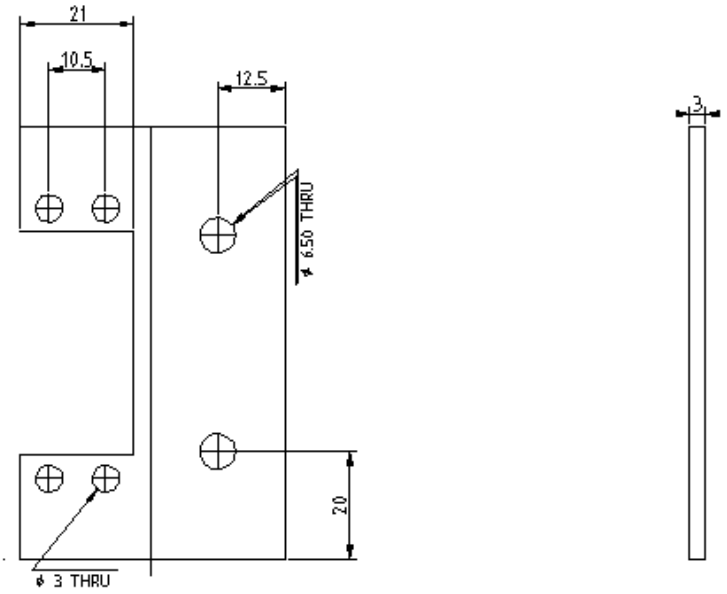
Bild 5.1: Roboter der AGAS

Anhang A

Konstruktionszeichnungen



Servohalterung
Maßstab: 1:1
Maßeinheit: mm



Anhang B

Aufbau Enwicklungs- und Softwareumgebung

Alle Programme wurden unter Linux und mit Ausnahme der Programme für den Microkontroller für Linux entwickelt. Als Compiler wurde *gcc* in der Version 4.1.2 verwendet und als Programmbibliothek zur Gestaltung der Grafischen Oberfläche wurde *QT* in der Version 4.2.3 eingesetzt.

B.1 Erkennung der Geräte

Der Laserscanner wird als USB Modem (CDC ACM) erkannt, das erforderliche Kernelmodul heißt *cdc-acm* bei Kernel 2.6.20 und ist in der Kernelkonfiguration unter `Device Drivers => USB Support => <M> USB Modem (CDC ACM) support` zu finden. Die Gerätedatei heißt `/dev/ttyACM0`. Das Crumb8-USB Modul benötigt das Kernelmodul *cp2101*, welches ab Kernelversion > 2.16.11 fester Bestandteil ist. Das Modul *ftdi_sio* wird von dem USB Programmierer benötigt. Beide sind in der Konfiguration unter `Device Drivers => USB Support => USB Serial Converter support ->` zu finden.

B.2 udev-rules

Udev[1] ist ein Programm (Systemdienst) das die Verwaltung des */dev* Verzeichnisses unter Linux übernimmt. Wenn neue Geräte angeschlossen werden, erzeugt udev dynamisch sogenannte Geräteknoten in dem */dev* Verzeichnis. Bei Geräten mit einem Seriell/USB Konverter sind das Geräteknoten der Form */dev/ttyUSBx*, wobei x eine fortlaufende Numerierung der Geräte ist. Die Reihenfolge der Numerierung ergibt sich aus der Reihenfolge, in der die Geräte erkannt werden. Auf dem Roboter der AGAS werden verschiedene Geräte mit solch einem Konverter eingesetzt. Udev erlaubt das Erstellen von Regeln, die dynamisch einen zweiten Geräteknoten erzeugen, dieser zweite Geräteknoten (Alias) ist ein symbolischer Link auf den richtigen Knoten. Durch das Erzeugen eines Aliases enfällt das lästige Ändern der Konfigurationsdateien und Programmdateien, da der symbolische Link immer gleich heißt und jeweils auf die richtige Gerätedatei zeigt. Die Regeln werden im Verzeichnis */etc/udev/rules.d/* in Dateien abgespeichert. Es wurde eine neue Datei *98-udev-robbie.rules* angelegt. Die erste Regel legt einen Alias */dev/laseruC* an, setzt die Rechte auf *0660* und führt nach dem Einstecken das Programm */usr/bin/SetPortFlags.sh /dev/%k* aus, welches einige Flags richtig setzt. Die zweite Regel ist für das Crumb8-Modul des Thermal Image Sensors. Da im Auslieferungszustand beide Module die gleiche Seriennummer hatten und so eine Unterscheidung per Regel nicht möglich war, wurde mit Hilfe der Anleitung[2] und Software[3] die Seriennummer der Module geändert.

B.3 Programmierung des Microkontrollers

Als Programmiereditor wurde Kate (K advanced text editor) benutzt, das Syntaxhighlighting für AVR-Assembler bietet. Die so erstellte .asm Datei wurde mit Hilfe von *AVRA*[4], einem Open Source Assembler für die Atmel Microkontrollerfamilie, übersetzt.

```
avra lasercontrol.asm
```

[1] Gutes Tutorial unter http://www.reactivated.net/writing_udev_rules.html

[2] http://www.silabs.com:80/public/documents/tpub_doc/anote/Microcontrollers/Interface/en/an144.pdf

[3] http://www.silabs.com:80/public/documents/software_doc/othersoftware/Microcontrollers/Interface/en/an144sw.zip

[4] http://avra.sourceforge.net/

Listing B.1: 98-udev-robbie.rules

```
#######################Laserscanner MicroController Device#########################
#
# Add the Alias devname /dev/laserUC and sets the Permissions to "0666"
#
# Setting Port Flags in /usr/bin/SetPortFlags.sh
#
Kernel=="ttyUSB*", ATTRS{product}=="CP2102␣USB␣to␣UART␣Bridge␣Controller", \
ATTRS{serial}=="0002", SYMLINK+="laserUC", MODE="0666", ACTION=="add", \
RUN+="/usr/bin/SetPortFlags.sh␣/dev/%k"
#
#
#
############################ThermalImageSensor MicroController Device###############
#
#
# Add the Alias devname /dev/thermalUC and sets the Permissions to "0666"
#
# Setting Port Flags in /usr/bin/SetPortFlags.sh
#
Kernel=="ttyUSB*", ATTRS{product}=="CP2102␣USB␣to␣UART␣Bridge␣Controller", \
ATTRS{serial}=="0003", SYMLINK+="thermalUC", MODE="0666", ACTION=="add", \
RUN+="/usr/bin/SetPortFlags.sh␣/dev/%k"
#
#
```

Listing B.2: SetPortFlags.sh

```
#!/bin/sh
#
#Author: Christian Delis
#
# Sets the Port Flags
#
DEVICE="$1"
stty --file="$DEVICE" -hupcl -opost -onlcr -iexten -echoe -echok -echoctl -echoke -echo
#
```

Zur Programmierung des *ATMEL* Microkontrollers in *AVR-Assembler* wurde das freie Programm *avrdude*[5] in Version 5.1 verwendet, da erst ab Version 5.0 das von dem Programmieradapter verwendete Protokoll *stk500v2* unterstützt wird. Die von *avra* erzeugte .hex (Intel Hex Format) wurde mit dem Programmieradapter und folgendem Befehl auf den Microkontroller übertragen. Die Software führt nach dem Programmieren einen Reset aus, so dass das neue Programm automatisch nach dem Schreibvorgang geladen wird.

```
avrdude -c stk500v2 -p m8 -P /dev/usbisp -b 115200 -V -U flash:w:lasercontrol.hex
```

Vor der ersten Benutzung wurden die Fusebits so geändert, dass der externe Quarz mit 14,7456 Mhz anstelle des internen 1 Mhz Quarzes benutzt wird. Folgender Befehl setzt die Fusebits für den externen Taktgeber:

```
avrdude -c stk500v2 -p m8 -P /dev/usbisp -b 115200 -V -U hfuse:w:0xc8:m -U lfuse:w:0xdf:m
```

B.4 Sonstige Software

Desweiteren wurde folgende Software eingesetzt:

Qcad Zur Erstellung der Konstruktionszeichnungen.

Inkscape Erstellung des Zustandsautomaten.

Gimp Konvertierung der Screenshots in das .eps Dateiformat.

Eagle Zur Gestaltung des Platinenlayouts.

[5] savannah.nongnu.org/projects/avrdude

Anhang C

Quelltexte

Listing C.1: lasercontrol.asm

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Author: Christian Delis <cdelis@uni-koblenz.de>
;;
;; AGAS Uni-Koblenz    <agas@uni-koblenz.de>
;;
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Includes
;;
.include "m8def.inc"                    ; include file for Atmel Atmega8
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;   Definition
;;
.def temp = r16                         ; temp register
.def temp1= r17                         ; temp register
.def pos= r18                           ; hold the current position/scan number
.def safe= r19                          ; register to safe the SREG
.def maxPos=r20                         ; scanCount, number of scans
.def stepL=r21                          ; Low Byte of the stepwidth inkrement
.def stepH=r22                          ; High Byte of the stepwith inkrement
.def rdata=r23                          ; Dataregister for Receive and Transfer
.def durationL=r26                      ; Low Byte - Duration of high signal in clocks
.def durationH=r27                      ; High Byte - Duration of high signal in clocks
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Communication Constant
```

```
;;
.equ REQUEST_HOLD_2D_POS ='P'
.equ REQUEST_VERSION ='v'
.equ REQUEST_3D_SCAN_CTRL ='S'
.equ REQUEST_STOP_2D = 'R'
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Constant
;;
.equ FCPU = 14745600                 ; CPU frequenz in Hz
.equ BAUD = 115200                   ; Baudrate for uart
.equ RAMSTART=0x0060 ;               ; Adress of ramstart
.equ UBRRVAL = FCPU/(BAUD*16)-1      ; calculate UBRRVAL
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Versionsinfo String
;;
version:                             ; String should be an odd number of char
          .db "Lasercontrol Program: Version 1.0",0

;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Macros
;;
.macro  mdelay                       ; delay macro
          ldi   r24, low( @0 - 8 )   ; max 65536 clocks => max 4.44 ms @14.7456 MHZ
          ldi   r25, high( @0 - 8 )  ; @0 first parameter
          sbiw  r24, 4
          brcc  pc - 1
          cpi   r24, 0xFD
          brcs  pc + 4
          breq  pc + 3
          cpi   r24, 0xFF
          breq  pc + 1
.endmacro

.macro msdelay                       ; delay for @0 msec
        mdelay ((FCPU * @0) / 1000)
.endmacro
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; interrupt vector
.org 0x000                           ; start execution after reset
          rjmp RESET
.org INT0addr                        ; interrupt adress for external interrupt 0
          rjmp int0_handler
.org URXCaddr                        ; Interrupt adress for UART RECEIVE
          rjmp int_rxc
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
RESET:                                  ; main routine

        ldi temp, LOW(RAMEND)           ; initialize Stackpointer
        out SPL, temp
        ldi temp, HIGH(RAMEND)
        out SPH, temp

                                        ; initialize UART Interface
        ldi temp, LOW(UBRRVAL)          ; set baudrate
        out UBRRL, temp
        ldi temp, HIGH(UBRRVAL)
        out UBRRH, temp


        ldi temp, (1<<URSEL)|(3<<UCSZ0) ; set Frame-Format: 8 Bit, 1 Stop bit
        out UCSRC, temp

        sbi UCSRB,TXEN                  ; enable UART transfer - TX
        sbi UCSRB, RXEN                 ; enable UART receive - RX

                                        ; Pin PD5 generates PWM
        ldi temp, (1<< PD5)             ; set data direction to output
        out DDRD, temp



mainloop:
        rcall uart_receive              ; receive control char
check_rx:                               ; send control char back
        sbis UCSRA,UDRE
        rjmp check_rx
        out UDR, rdata

        cpi rdata, REQUEST_3D_SCAN_CTRL ; if REQUEST_3D_SCAN_CTRL initiate new Scan
        breq newscan
        cpi rdata,REQUEST_HOLD_2D_POS   ; if REQUEST_HOLD_2D_POS hold at 2DPos
        breq pos2D
        cpi rdata, REQUEST_VERSION      ; if REQUEST_VERSION send Version Info
        breq sndversion
        jmp mainloop                    ; end mainloop

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; This routine executes the 3D Scan Control
newscan:

        ldi YH,HIGH(RAMSTART)           ; initialize rampointer
```

```
125             ldi YL,LOW(RAMSTART)
126
127             rcall uart_receive
128             mov durationL, rdata        ; store start pos, duration of highsignal, low Byte
129             rcall uart_receive
130             mov durationH, rdata        ; store start pos, duration of highsignal, high Byte
131             rcall uart_receive
132             mov stepL,rdata             ; store stepwidth, low byte
133             rcall uart_receive
134             mov stepH,rdata             ; store stepwidth, high byte
135             rcall uart_receive
136             mov maxPos, rdata           ; store scanCount, number of scans
137
138 maxpos_tx:
139             sbis UCSRA,UDRE             ; send maxPos back, for control
140             rjmp maxpos_tx
141             out UDR, maxPos
142
143             ldi temp, 0
144 bresen:
145
146             rcall uart_receive
147             st Y+, rdata                ; store correction at Rampos Y and inc rampointer
148
149 transmit:
150             sbis UCSRA,UDRE             ; send correction back
151             rjmp transmit
152             out UDR, rdata
153
154             inc temp
155             cp temp, maxPos             ; repeat until not maxPos stepcorrections received
156             brlo bresen
157
158 count_tx:
159             sbis UCSRA,UDRE
160             rjmp count_tx
161             out UDR, temp               ; send stepcorrection back
162
163
164             ldi YH,HIGH(RAMSTART)       ; set Y-Pointer back to start for reading
165             ldi YL,LOW(RAMSTART)
166
167             rcall resetservo            ; move Servo to start Position
168
169             ldi pos, 0                  ; set initialvalue
170
171
172 sync_tx:                                ; sync with pc, this stops the pc program
```

```
        sbis UCSRA,UDRE             ; if not, the pc requests a 2D-scan from scanner
        rjmp sync_tx                ; before the servo is at the start Position
        out UDR, temp


        ldi temp, 0b00000010        ; configure external interrupt 0 (INT0)
        out MCUCR, temp             ; interrupt when falling edge

        ldi temp, 0b01000000        ; activate INT0
        out GIMSK, temp


loop:
        cli                         ; deactivate global interrupt!
        ;no interrupt here, period of high signal has to be complied
        ldi temp, 0b00100000
        out PORTD, temp             ; set PD5 High
        rcall delay_sig_high_start  ; wait duration
        cbi PORTD, PD5              ; set PD5 to Low
        sei                         ; activate interrupt!
        rcall delayms20             ; PD5 for 20ms low
        cp pos, maxPos              ; reached maxPos?
        brne loop
        cli                         ; deactivate interrupts


transmit_durH:
        sbis UCSRA,UDRE             ; send reached duration back to pc
        rjmp transmit_durH          ; for control
        out UDR, durationH
transmit_durL:
        sbis UCSRA,UDRE
        rjmp transmit_durL
        out UDR, durationL

        ;ldi rdata, 0               ; overwrite rdata !!!!!!!!!!!!!!!!!!!!!
        ldi maxPos, 0               ; clean up
        jmp mainloop                ; scan finished, jump back
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; This routine moves the servo to the start position
;;
resetservo:
        ldi temp1, 0
reset_:
        ldi temp, 0b00100000
        out PORTD, temp             ; set PD5 High
        rcall delay_sig_high_start  ; wait
        cbi PORTD, PD5              ; set PD5 Low
```

```
            rcall delayms20              ; 20 ms warten
            inc temp1
            cpi temp1, 25                ; 300ms for 180° => 25 runs to move to start Pos
            brne reset_
            ret
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Delaying for start Position
;;
delay_sig_high_start:
            mov   r24, durationL
            mov   r25, durationH
            sbiw  r24, 4
            brcc  pc - 1
            cpi   r24, 0xFD
            brcs  pc + 4
            breq  pc + 3
            cpi   r24, 0xFF
            breq  pc + 1
            ret
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Delay 20 ms
;;
delayms20:              ; 5*4 = 20 :-)
            msdelay 4
            msdelay 4
            msdelay 4
            msdelay 4
            msdelay 4
            ret
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; This routine sends the position pos via uart
uart_transmit:
            sbis UCSRA,UDRE
            rjmp uart_transmit
            out UDR, pos
            ret
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; This is the external interrupt 0 handler, it adds the stepwidth to the duration
;;
int0_handler:                            ; interrupt service routine
            in safe, SREG                ; safe SREG to stack
            push safe                    ;
            rcall uart_transmit          ; send pos
            ld temp, Y+                  ; load stepcorrection (bresenham) to temp
```

```
            sbrc temp, 0                ; if not 0 add 1 to duration
            adiw durationL, 1           ;
            add durationL, stepL        ; increase duration => 147,456 is 0.01 ms
            adc durationH, stepH        ;
            inc pos                     ; increment pos
            pop safe                    ; restore SREG from stack
            out SREG, safe
            reti
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; This routine receives one byte and stores it in register rdata
;;
uart_receive:
            sbis UCSRA, RXC             ; wait until one byte was received
            rjmp uart_receive
            in rdata, UDR
            ret
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; This routine holds the Servo(Scanner) at the 2D scan Position
;;
pos2D:
            rcall uart_receive          ; receive the 2D Scan Position in clocks
            mov durationL, rdata                    ;
            rcall uart_receive
            mov durationH, rdata
            check_rx1:
            sbis UCSRA,UDRE             ; send back high byte of 2d scan pos in clocks
            rjmp check_rx1
            out UDR, rdata
            sbi UCSRB, RXCIE            ; enable receive interrupt
nextsig:                                ; generate pwm
            cli
            ldi temp, 0b00100000        ;
            out PORTD, temp             ; set PD5 High
            rcall delay_sig_high_start  ; wait duration of high
            cbi PORTD, PD5              ; set PD5 Low
            sei
            rcall delayms20             ; wait 20 ms
            cpi temp1, REQUEST_STOP_2D  ; check if interrupt has occured
            brne nextsig
            mov rdata, temp1            ; send control char back
check_rx2:
            sbis UCSRA,UDRE             ;
            rjmp check_rx2
            out UDR, rdata
            jmp mainloop
;;
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; This routine is the receive byte interrupt routine,
;; it is used to cancel the holding of 2D Pos
;;
int_rxc:
          in rdata, UDR
          mov temp1, rdata          ; copy char to temp1, so we can check
          cbi UCSRB, RXCIE          ; disable receive interrupt
          reti
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; This routine sends the Programm Version
;;
sndversion:
        ldi ZL, LOW(version*2)      ; load StringAddress to Z Pointer
        ldi ZH, HIGH(version*2)     ;
print:
        lpm rdata, Z+               ; load byte, addressed by Z, to rdata
        tst rdata                   ; test if 0 => String end
        breq print_end              ; if 0, end of output
        mov r16, r0                 ; Inhalt von R0 nach R16 kopieren
        rcall sendchar              ; UART-Sendefunktion aufrufen
        adiw ZL, 1                  ; Adresse des Z-Pointers um 1 erhöhen
        rjmp print                  ; wieder zum Anfang springen
print_end:
        ret

        ldi rdata, 10               ; 10 == LF
        rcall sendchar              ; 13 == CR
        ldi rdata, 13
        rcall sendchar              ; send line break

sendchar:
          sbis UCSRA,UDRE           ; send character
          rjmp sendchar
          out UDR, rdata
          jmp mainloop
```

Listing C.2: datareader.cpp

```
/*************************************************************************
 *  Filename: datareader.cpp
 *
 * Author: Christian Delis <cdelis@uni-koblenz.de>
 *
 *  (C) 2006 AG Aktives Sehen <agas@uni-koblenz.de>
 *          Universitaet Koblenz-Landau
 *************************************************************************/
```

```
9
10
11 #include "datareader.h"
12 #include <iostream>
13 #include <QString>
14 #include <cmath>
15 #include <QFile>
16 #include <QTextStream>
17 #include <cmath>
18
19 #include "config.h"
20
21 //bitrate for laserscanner, it is fixed, so don't change it!
22 #define LASER_DEVICE_BITRATE B19200
23 //bitrate for Microcontroller, it is in the uC programmed so don't change it!
24 #define MICROCONTROLLER_BITRATE B115200
25 //use a min cloud size, so we haven't to resize the vector so often
26 #define MIN_CLOUD_SIZE 50000
27 #define ANGLE_RESOLUTION  360.0 / 1024.0
28 #define DEGREE_TO_RADIANT  M_PI / 180.0
29 #define PHICORRECT 0
30 //liefert Bit 8 bis 15
31 #define HIGHBYTE(x) ((x & 0xFF00) >>8)
32 //liefert bit 0 bis 7
33 #define LOWBYTE(x) ((x) & 0xFF)
34 #define REQUEST_HOLD_2D_POS  'P'
35 #define REQUEST_VERSION  'v'
36 #define REQUEST_3D_SCAN_CTRL  'S'
37 #define REQUEST_STOP_2D  'R'
38
39
40
41 DataReader::DataReader()
42 {
43   m_uCDevice = "/dev/laseruC";
44   m_scanDevice = "/dev/ttyACM0";
45   m_readingMutex = false;
46   m_scannerFD = NULL;
47   m_uCFD = NULL;
48   m_devicesConnected = false;
49   m_longRangeMode = false;
50 }
51
52 Laserscan3D* DataReader::getLaserScanObject(){
53 if(m_myScan){return m_myScan;}
54 return NULL;
55 }
56
```

```
57
58 DataReader::~DataReader()
59 {
60 if(m_myScan){delete m_myScan;}
61 }
62
63 bool DataReader::isDataReading(){  return m_readingMutex;}
64
65 void DataReader::holdPos(int holdpos){
66   if(!m_devicesConnected){
67       connectDevices();
68     }
69
70   if(!m_devicesConnected){
71       std::cout << "ERROR: Connection to devices fail! Did you connect them?" << std::
              endl;
72       m_readingMutex=false;
73       return;
74     }
75
76   int uCControl=REQUEST_HOLD_2D_POS;
77
78   if(!fputc(uCControl,m_uCFD)){
79      std::cout << "ERROR: Could not write uCControl to uC!" << std::endl;
80    }
81   if( holdpos < DEFAULTSTART_POS ||holdpos > DEFAULTEND_POS){
82
83    std::cout << "ERROR: holdpos is out of range!" << std::endl;
84    return;
85   }
86
87   std::cout << " uCControl byte send" << fgetc(m_uCFD)<<" to uC!" << std::endl;
88   int holdposL=LOWBYTE(holdpos);
89   int holdposH=HIGHBYTE(holdpos);
90
91   if(!fputc(holdposL,m_uCFD)){
92      std::cout << "ERROR: Could not write holdposL to uC!" << std::endl;
93    }
94
95   if(!fputc(holdposH,m_uCFD)){
96      std::cout << "ERROR: Could not write holdposH to uC!" << std::endl;
97    }
98 std::cout << " Am I in gopos subroutine? " << fgetc(m_uCFD)*256<<" \n" << std::endl;
99  m_holdPos=true;
100 }
101
102 void DataReader::stopHoldPos(){
103
```

```
104   int uCControl=REQUEST_STOP_2D;
105   if(!fputc(uCControl,m_uCFD)){
106       std::cout << "ERROR: Could not write stopHoldPos to uC!" << std::endl;
107     }
108 std::cout << " stopHold, return from gopos" << fgetc(m_uCFD)<<" to uC!" << std::endl;
109 m_holdPos=false;
110 }
111
112  void DataReader::getScanFromLaser(int scancount,int startPos, int endPos, int* numPoints
        , float** xCoords, float** yCoords, float** zCoords){
113
114     if(scancount< MIN_SCANCOUNT || scancount > MAX_SCANCOUNT){return;}
115     if(startPos < MINSTART_POS || endPos > MAXSTART_POS){return;}
116    // first stop holdpos?
117    if(m_holdPos){stopHoldPos();}
118
119    int scanCount=LOWBYTE(scancount);
120    int stepwidth= (int) ((endPos - startPos)/scanCount);
121    int durationL=LOWBYTE(startPos);
122    int durationH=HIGHBYTE(startPos);
123    int stepL=LOWBYTE(stepwidth);
124    int stepH=HIGHBYTE(stepwidth);
125    int uCControl=REQUEST_3D_SCAN_CTRL;
126    int * stepCorrection;
127
128    std::cout << "scanCount: " << scanCount << std::endl;
129    std::cout << "stepwidth: " << stepwidth << std::endl;
130    std::cout << "duration: " << ((durationH*256)+ durationL) << std::endl;
131    std::cout << "Step: " << ((stepH * 256) + stepL) << std::endl;
132
133    int pos;
134    int * scan;
135    //double res= 180.0 / scanCount;
136    int scanSteps = 682;
137    //double correction=2.0;
138    int* wholeScan = new int[scanCount * scanSteps];
139    //are we already reading?
140    if(m_readingMutex){
141      return;
142    }
143    // lock reading
144    m_readingMutex = true;
145
146    if(!m_devicesConnected){
147
148      connectDevices();
149    }
150
```

```
151     if(!m_devicesConnected){
152       std::cout << "ERROR: Connection to devices fail! Did you connect them?" << std::endl
              ;
153       m_readingMutex=false;
154       return;
155     }
156 
157      if(!fputc(uCControl,m_uCFD)){
158       std::cout << "ERROR: Could not write uCControl to uC!" << std::endl;
159     }
160 
161  std::cout << " uCControl byte send" << fgetc(m_uCFD)<<" to uC!" << std::endl;
162 
163     if(!fputc(durationL,m_uCFD)){
164       std::cout << "ERROR: Could not write durationL to uC!" << std::endl;
165     }
166     if(!fputc(durationH,m_uCFD)){
167       std::cout << "ERROR: Could not write durationH to uC!" << std::endl;
168     }
169     if(!fputc(stepL,m_uCFD)){
170       std::cout << "ERROR: Could not write stepL to uC!" << std::endl;
171     }
172     if((fputc(stepH,m_uCFD))== EOF){
173       std::cout << "ERROR: Could not write stepH to uC!" << std::endl;
174     }
175 
176     if(!fputc(scanCount,m_uCFD)){
177       std::cout << "ERROR: Could not write scanCount to uC!" << std::endl;
178     }
179     std::cout << " Successful transmit scanCount " << fgetc(m_uCFD)<<" to uC!" << std::
          endl;
180 
181     bresen(scanCount,startPos, endPos, stepCorrection );
182 //     for(int i=0; i < scanCount; i++){
183 //       std::cout << "stepCorrection["<< i <<"] " << stepCorrection[i] << std::endl;
184 //     }
185   for(int i=0; i < scanCount; i++){
186     if(fputc(stepCorrection[i],m_uCFD)== EOF){
187       std::cout << "ERROR: Could not write stepCorrection["<< i <<"] " << stepCorrection[
              i]<<" to uC!" << std::endl;
188     } else{
189 
190       std::cout << "Write: stepCorrection["<< i <<"] " << stepCorrection[i]<<" Received: "
              << fgetc(m_uCFD)<< std::endl;
191     }
192     //usleep(500);
193     //
194 
```

```
195     }
196     //std::cout << "after stepcorrection transmission" << std::endl;
197    //std::cout <<fputc(scanCount,m_uCFD)<< std::endl;
198     std::cout << "␣number␣of␣stepCorrection␣successfully␣transmited␣" << fgetc(m_uCFD)<<"␣
            to␣uC!" << std::endl;
199     //std::cout << std::endl;
200     //std::cout << "scanCount: " << scanCount << "\n" << std::endl;
201     //std::cout << std::endl;
202     // Obtain data for readybyte scan, this should be one complete 3d scan
203
204     fgetc(m_uCFD);          //no other purpose, just for sync
205     for(int i=0;i<scanCount;++i)
206     {
207       std::cout << "Trying␣scan␣#" << i << std::endl;
208
209       // Obtain all step (0-768) distance data
210       scan=urgGetScan(0,768,1);
211       if(scan==NULL){
212
213       std::cout << "ERROR:␣Could␣not␣get␣2D␣Scan!" << i << std::endl;
214       m_readingMutex=false;
215       return;
216       }
217       //get Position from uC
218       pos=fgetc(m_uCFD);
219       if(pos == EOF){
220          std::cout << "ERROR:␣Could␣not␣get␣Position␣from␣uC!" << i << std::endl;
221          m_readingMutex=false;
222         return;
223
224       }
225
226       if(scan!=NULL)
227       {
228
229
230         for(int pt=44;pt<726;pt++)
231         {
232           wholeScan[i*scanSteps + pt - 44 ] = scan[pt];
233         }
234         free(scan);
235       }
236       else
237       {
238         std::cout << "scan␣#" << i << "␣failed" << std::endl;
239       }
240     }
241
```

```
242     std::cout << "Endpos: " << ((fgetc(m_uCFD)*256) + fgetc(m_uCFD)) << std::endl;
243
244     //float t_0 = 120.0 / 180.0 * M_PI; //false
245     float t_0 = (135 - (44*ANGLE_RESOLUTION)) / 180.0 * M_PI;
246     //float TR = -240 / 180.0 * M_PI; //false
247     float TR = -(682*ANGLE_RESOLUTION) / 180.0 * M_PI;
248     float p_0 = - M_PI / 2.0;
249     float PR = M_PI;
250
251     m_myScan = new Laserscan3D(t_0,TR,scanSteps,p_0,PR,scanCount,wholeScan);
252     m_myScan->get3DPoints(numPoints, xCoords,yCoords, zCoords);
253     //unlock reading
254     m_readingMutex = false;
255   }
256
257
258 void DataReader::connectDevices(){
259
260   //set up scanner
261   m_scannerFD =fopen(m_scanDevice,"w+b");
262   if(m_scannerFD==NULL){
263     std::cout << std::endl;
264     std::cout << "ERROR: Could not open ScanDevice! Is Scanner connected and is
          Kernelmodul \"cdc_acm\" loaded?" << std::endl;
265     std::cout << std::endl;
266     return;
267   }
268
269   if(!setupSerial(fileno(m_scannerFD),LASER_DEVICE_BITRATE)){
270
271     std::cout << "ERROR: Setting up Serialattributes to "<< m_scanDevice<<" failed!" <<
          std::endl;
272     fclose(m_scannerFD);
273   }
274
275
276   // set up Microcontroller
277   m_uCFD =fopen(m_uCDevice,"w+b");
278   if(m_uCFD==NULL){
279
280     std::cout << std::endl;
281     std::cout << "ERROR: Could not open uC Device! Is MicroController connected and is
          Kernelmodul \"cp2101\" loaded?" << std::endl;
282     std::cout << std::endl;
283     return;
284   }
285
286   if(!setupSerial(fileno(m_uCFD), MICROCONTROLLER_BITRATE)){
```

```
287
288     std::cout << "ERROR: Setting up Serialattributes to "<< m_uCDevice <<" fail!" << std
            ::endl;
289     fclose(m_scannerFD);
290     fclose(m_uCFD);
291   }
292   m_devicesConnected=true;
293
294 }
295
296 void DataReader::switchScanRange(){
297
298 char req[16];
299 char line[80];
300 int i;
301 int controlcode=0;
302 //controlcode = 1 for longrange Mode otherwise controlcode =0 for non Longrange mode
303 if(m_longRangeMode){
304     controlcode =1;
305   }
306
307 sprintf(req,"E%01d\n",controlcode);
308 i=fwrite(req,sizeof(char),strlen(req),m_scannerFD);
309 if(i==0){
310   printf("ERROR: Switching to over 4096 Mode has failed!\n");
311   printf("fwrite return zero!\n");
312   return;
313 }
314
315   if(fgets(line,sizeof(line),m_scannerFD)==NULL){
316   printf("ERROR: Could not read from scanner!\n");
317   return;
318   }
319   //test if echo back is ok
320   if(strcmp(req,line)!=0)
321     {
322       printf("ERROR: Echo back is invalid!\n");
323       // invalid
324       return;
325     }
326
327    //read next line
328    if(fgets(line,sizeof(line),m_scannerFD)==NULL){
329     printf("ERROR: Could not read from scanner!\n");
330       return;
331     }
332    //test if status is ok => 0 if it was successful
333    if(strcmp("0\n",line)!=0)
```

```
334     {
335       printf("ERROR: Switching to over 4096 Mode has failed!\n");
336       printf("ERROR: Changing Mode has failed");
337      //invalid
338       return;
339      }
340    //read next line
341    if(fgets(line,sizeof(line),m_scannerFD)==NULL){
342     printf("ERROR: No LF at End of Echo back!\n");
343       return;
344     }
345
346    printf("CHANGING MODE WAS SUCCESSFUL\n");
347    m_longRangeMode=true;
348    return;
349 }
350
351
352
353 int*  DataReader::urgGetScan( int aStart, int aEnd, int aSkip)
354 {
355   //function taken from the example program of the hoyuko homepage
356   char req[16];
357   int *scan,*pt;
358   char line[80];
359   int i;
360
361   sprintf(req,"G%03d%03d%02d\n",aStart,aEnd,aSkip);
362   i=fwrite(req,sizeof(char),strlen(req),m_scannerFD);
363   if(i==0)
364     return NULL;
365   // echo-back
366   if(fgets(line,sizeof(line),m_scannerFD)==NULL)
367     return NULL;
368   if(strcmp(req,line)!=0)
369   {
370       // invalid
371     return NULL;
372   }
373
374   // status
375   if(fgets(line,sizeof(line),m_scannerFD)==NULL)
376     return NULL;
377   if(strcmp("0\n",line)!=0)
378   {
379       //invalid
380     return NULL;
381   }
```

```
382   // result
383   scan=(int*)malloc(sizeof(int)*((aEnd-aStart)/aSkip+1));
384   pt=scan;
385   while(1)
386   {
387     if(fgets(line,sizeof(line),m_scannerFD)==NULL)
388     {
389       free(scan);
390       return NULL;
391     }
392     if(!strcmp("\n",line))
393     {
394           //end of result
395       break;
396     }
397     for( unsigned int i=0;i<strlen(line)-1;++i,++pt)
398     {
399           // Note: It is assumed that received data is complete
400           //      (Problem will occour if data is incomplete)
401       *pt=((line[i] - 0x30)<<6) + line[++i] - 0x30;
402     }
403   }
404   return scan;
405 }
406
407 bool DataReader::setupSerial(int aFd, speed_t speed){
408
409     struct termios tio;
410      if(speed==0){
411
412       std::cout << "ERROR: Invalid Bitrate, set B115200 for uC or B19200 for Laserscanner
              !\n" << std::endl;
413       return false;  // invalid bitrate
414      }
415    //save old termios
416    tcgetattr(aFd,&m_Oldtio);
417
418    tcgetattr(aFd,&tio);
419
420    cfsetispeed(&tio,speed); // bitrate for input
421    cfsetospeed(&tio,speed); // bitrate for output
422
423    tio.c_cflag=(tio.c_cflag & ~CSIZE) | CS8; // data bits = 8bit
424
425    tio.c_iflag&= ~( BRKINT | ICRNL | ISTRIP );
426    tio.c_iflag&= ~ IXON;     // no XON/XOFF
427    tio.c_cflag&= ~PARENB;    // no parity
428    tio.c_cflag&= ~CRTSCTS;   // no CTS/RTS
```

```
429     tio.c_cflag&= ~CSTOPB;    // stop bit = 1bit
430
431     //OUTPUT
432     tio.c_oflag &= ~OPOST;
433
434    // Other
435     tio.c_lflag &= ~( ISIG | ICANON | ECHO );
436
437    // Commit
438      if(tcsetattr(aFd,TCSADRAIN,&tio)==0){ return true;     }
439      //fclose(aFd);
440      return false;
441
442 }
443
444
445 void DataReader::getScanFromFile(QString filename, int* numPoints, float **xCoords, float
       ** yCoords, float** zCoords){
446
447    //valid filename
448    if(filename.isEmpty()){
449      std::cout << "ERROR: Given Filename is invalid!" << std::endl;
450      return;
451      }
452     m_myScan=new Laserscan3D(filename.toAscii());
453     m_myScan->get3DPoints(numPoints, xCoords,yCoords, zCoords);
454    }
455
456
457    void DataReader::saveScanToFile(QString filename, int* numPoints, float** xCoords,
         float** yCoords, float** zCoords){
458      QFile file( filename );
459      if ( !file.open( QIODevice::WriteOnly ) ) {
460        std::cout <<"ERROR: Could not open File to write data" << std::endl;
461        return;
462      }
463
464      QTextStream textstream( &file );
465
466      if(!xCoords || !yCoords || !zCoords){
467        std::cout <<"ERROR: No Data to save!" << std::endl;
468        return;
469      }
470
471      for(int count=0; count < *numPoints; count++ ){
472        textstream << xCoords[count] << " " <<  yCoords[count]<< " " << zCoords[count];
473        endl(textstream);
474      }
```

```
475     std::cout <<"Saved " << *numPoints <<" Points!"<<  std::endl;
476     file.close();
477 }
478
479 void DataReader::saveScanToL3DFile(QString filename){
480   if(!m_myScan){
481   std::cout <<"ERROR: Writing to file failed!" << std::endl;
482   }
483
484   if(!m_myScan->writeToFile(filename.toAscii())){
485   std::cout <<"ERROR: Writing to file failed!" << std::endl;
486   }
487 }
488
489 void DataReader::bresen(int scanCount, int starty, int endy, int*& stepCorrection){
490  stepCorrection= new int[scanCount];
491 // sinnlose werte erstmal
492 for(int i = 0; i < scanCount; i++){
493 stepCorrection[i]=0;
494 }
495
496
497 double b;
498 //modf (cmath)  0,34= modf(2.34 , b);     => b = 2
499 float m=(float) modf( (endy - starty) /((double)scanCount), &b);
500 std::cout << "Gradient m=" << m << std::endl;
501
502
503 int sy=0;
504 int ey=(int)(m*scanCount);
505
506 int dx,dy;
507 dx = scanCount;
508 dy = ey - sy;
509
510 int eps=0;
511 int y=0;
512
513 for ( int x = 0; x < scanCount; x++ )  {
514         eps += dy;
515         if ( (eps << 1) >= dx )  {
516           stepCorrection[x]=1;
517           y++;
518           eps -= dx;
519         }
520       }
521
522 }
```

```
523
524 void DataReader::restorePortSettings(){
525 if (tcsetattr(fileno(m_uCFD),TCSADRAIN,&m_Oldtio) != 0) {
526     std::cout << "ERROR: DataReader::restorePortSettings(): tcsetattr() failed! Old port
          settings could not be restored!" << std::endl;
527   }
528 }
```

Literaturverzeichnis

[cru07] http://www.chip45.com/index.pl?page=Crumb8-USB&lang=de&tax=1&dest=0. (2007). papers/datasheets/microcontroller/IS_Crumb8-USB_060226.pdf

[hok06] http://www.hokuyo-aut.jp/products/urg/urg.htm. (2006). http://www.hokuyo-aut.jp/products/urg/urg.htm

[Nüc06] NÜCHTER, Andreas: *Semantische dreidimensionale Karten für autonome mobile Roboter*, Rheinische Friedrich-Wilhelms-Universität Bonn, Diss., 2006. https://www.uni-koblenz.de/~agas/Pool/Nuechter2006SDK.pdf

[PDMP06] PELLENZ, Johannes ; DELIS, Christian ; MIHAILIDIS, Ioannis ; PAULUS, Dietrich: Low-Cost 3D-Laserscanner für mobile Systeme im RoboCup Rescue Wettbewerb. In: *9. Anwendungsbezogener Workshop zur Erfassung, Modellierung, Verarbeitung und Auswertung von 3D-Daten* (2006)

[SD03] STEINHAUS, Peter ; DILLMANN, Rüdiger: Aufbau und Modellierung des RoSi Scanners zur 3D-Tiefenbildakquisition. In: *Autonome Mobile Systeme 2003*, Springer, 2003

[SLNH01] SURMANN, Hartmut ; LINGEMANN, Kai ; NÜCHTER, Andreas ; HERTZBERG, Joachim: *Aufbau eines 3D-Laserscanners für autonome mobile Roboter*. http://publica.fhg.de/eprints/B-73174.pdf,https://www.uni-koblenz.de/~agas/Pool/Surmann2001AE3.pdf. Version: 2001

[TMD+06] THRUN, Sebastian ; MONTEMERLO, Michael ; DAHLKAMP, H. ; STAVENS, D. ; ARON, A. ; DIEBEL, J. ; FONG, P. ; GALE, J. ; HALPENNY, M. ; HOFFMANN, G. ; LAU, K. ; OAKLEY, C. ; PALATUCCI, M. ; PRATT, V. ; STANG, P. ; STROHBAND, S. ; DUPONT, C. ; JENDROSSEK, L.-E. ; KOELEN, C. ; MARKEY, C. ; RUMMEL, C. ; NIEKERK, J. van ; JENSEN, E. ; ALESSANDRINI, P. ; BRADSKI, G. ; DAVIES, B. ; ETTINGER, S. ; KAEHLER, A. ; NEFIAN, A. ; MAHONEY, P.: Winning the DARPA Grand Challenge. In: *Journal of field Robotics* (2006). – accepted for publication

[URG04] *URG Series Communication Protocol Specification*. : *URG Series Communication Protocol Specification*, 2004. `papers/datasheets/laserscanner/URGCommSpec.pdf`

[URG05] *Scanning Laser Range Finder URG-04LX Specifications*. : *Scanning Laser Range Finder URG-04LX Specifications*, 2005. `papers/datasheets/laserscanner/URGSpecificaton.pdf`

[WW03] WULF, O. ; WAGNER, B.: Fast 3D-Scanning Methods for Laser Measurement Systems. In: *14th International Conference on Control Systems and Computer Science* (2003). `file:///lab/as/Docs/Protected/Wulf2003F3M.pdf,fish://serres/lab/as/Docs/Protected/Wulf2003F3M.pdf`