

MASTER THESIS

zur Erlangung des akademischen Grades
„Master of Science in Engineering“
im Studiengang Industrielle Elektronik

Aufbau eines automatisierten Mess- und Auswertesystems zur Bestimmung der Bestrahlungsstärkeverteilung in einem stationären Sonnensimulator

Ausgeführt von: Thomas Schmatz BSc

Personenkennzeichen: 1010300002

1. BegutachterIn: DI Bernhard Kubicek

2. BegutachterIn: DI (FH) Thomas Krametz

Wien, 9. Juli 2012

Eidesstattliche Erklärung

„Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbständig angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher weder in gleicher noch in ähnlicher Form einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht. Ich versichere, dass die abgegebene Version jener im Uploadtool entspricht.“

Ort, Datum

Unterschrift

Kurzfassung

Die Arbeit umfasst die Planung und Realisierung eines selbstfahrenden Messroboters, der die Bestrahlungsstärkeverteilung in der Prüfebene eines stationären Sonnensimulators erfasst. Neben den Einstrahlungsdaten werden weitere zusätzliche Informationen über die Umgebungsbedingungen im Prüfkanal aufgezeichnet. Bei der Umsetzung wurden Rapid-Prototyping Techniken (3D-Druck, Platinenfräse und Lasercutter) eingesetzt. Behandelt werden theoretische Grundlagen und normative Anforderungen an stationäre Sonnensimulatoren, sowie Messunsicherheitsberechnungen und Validierung des Gesamtsystems.

Schlagwörter: Sonnensimulator, Messroboter, Bestrahlungsstärke

Abstract

This paper discripes the planning and the realisation of a autonomous measurement robot. The robot measures the irradiance in a test plane of a continous solar simulator.

Keywords: Measurement Robot, Measurment of Solar Radiation, Arduino, Mecanum Wheel,
Keyword 5

Danksagung

Ich danke meinen Eltern für die Unterstützung und Geduld, die sie während des Studiums aufgebracht haben. Ich danke meinen Hochschulbetreuer für die umfangreiche Betreuung. Ich danke meinen Firmenbetreuer Thomas Krametz für die Zeit die er sich genommen hat. Und für problemlose Verlängerung meines Praktikums und der Diplomarbeit schulde ich Herrn Wolfgang Hribernik Wolfgang großen Dank.

Inhaltsverzeichnis

1. Aufgabenstellung	1
1.1. Stationäre Photovoltaik Sonnensimulatoren	1
1.2. Sonnensimulator Aufbau	1
1.3. Normative Anforderungen	2
1.4. Theorie Referenzzelle	4
2. Entwicklungsprozess	8
2.1. Hardwaredesign und Komponenten	8
2.1.1. Die Mecanum-Plattform	8
2.1.2. Chassis	11
2.2. Steuerungselektronik	11
2.2.1. Motorensteuerung	14
2.2.2. Optische Sensorik	14
2.2.3. Spannungversorgung	17
2.2.4. Mikrokontroller	18
2.2.5. Temperatursensoren	18
2.3. Softwareentwicklung	18
2.3.1. Entwicklungsumgebung	20
2.3.2. Auswertung der optischen Sensoren	20
2.3.3. Auswertung ADCs	21
2.3.4. Programmablauf	21
3. Kalibration	25
3.1. Temperatursensoren	25
3.2. Messzelle	25
3.3. Strommessung	29
3.4. Thermische Stabilität der Temperaturmessung	29
4. Messung	32
4.1. Messaufbau	32
4.1.1. Platten	32
4.1.2. Ablauf	33
4.2. Auswertung	34
4.3. Vermessung der Ausleuchtung einzelner Lampen	35
4.4. Schlussfolgerung	42
Literaturverzeichnis	43
Abbildungsverzeichnis	45

Tabellenverzeichnis	46
Abkürzungsverzeichnis	47
A. Sourcecode Arduino	48
B. Sourcecode Auswertung	63

1. Aufgabenstellung

Dieses Kapitel beschreibt die Notwendigkeit von Messungen im stationären Sonnensimulator, den Aufbau eines solchen Simulators, sowie dessen normativen Anforderungen. Weiters wird die Funktion eines Photovoltaik-Zelle beschrieben.

1.1. Stationäre Photovoltaik Sonnensimulatoren

Photovoltaik-Sonnensimulatoren werden zur elektrischen Charakterisierung und Prüfung von Photovoltaik-Modulen verwendet. Der wesentliche Vorteil liegt darin, dass die Durchführung unter reproduzierbaren Umweltbedingungen (Bestrahlungsstärke, spektrale Zusammensetzung des einfallenden Lichtes und die Prüfgutstemperatur) ermöglicht wird. Grundsätzlich werden bei konventionellen Systemen gepulste und stationäre Simulatoren, welche kontinuierlich leuchten, unterschieden. Erstere brauchen weniger Energie und heizen die Messobjekte nicht auf, und werden für die Leistungsmessung von Zellen und Modulen verwendet. Allerdings ist Blitzlänge bauartbedingt auf wenige Millisekunden begrenzt. Stationäre Simulatoren werden für Voralterung von Modulen verwendet. Im Allgemeinen werden Messergebnisse auf Standard-Prüfbedingungen (STC) bezogen, die durch 1000W/m^2 , 25°C und AM1.5 definiert ist. Die Air Mass (AM) beschreibt das Verhältnis des Weges des einfallenden Sonnenlichtes durch die Atmosphäre bezogen auf senkrechten Einfall. Je länger der Weg des Lichtes durch die Atmosphäre desto größer ist der Einfluss durch Streuung, Reflexion und Absorption, was sowohl die Stärke der Einstrahlung als auch die spektrale Zusammensetzung des Sonnenlichtes vom Sonnenstand abhängig macht.

$$AM = \frac{L}{L_0} \approx \frac{1}{\cos z} \quad (1.1)$$

Wobei L die Länge des Weges des Sonnenlichtes durch die Atmosphäre, L_0 die Weglänge bei senrekreten Einfall und z der Zenitwinkel des Sonnenstandes in Grad ist ([1] S. 30). Die spektrale Verteilung des Referenzsonnenspektrums AM1.5 (siehe Abbildung 1.1) ist durch die Norm IEC 60904-3 [2] festgelegt. Das Spektrum des Sonnenlichtes im Weltall AM0 entspricht in etwa der thermische Strahlung eines schwarzen Körpers mit 5250°C . Am Markt erhältliche Sonnensimulatoren für Modulgröße ($1,6\text{ m}^2$) können das Spektrum nicht verändern, wegen der verwendeten Lampen, sind also für einen Sonnenstand beschränkt. Es gibt Sonnensimulatoren auf LED-Basis. Das gewünschte Spektrum wird dort mit vielen LEDs unterschiedlicher Wellenlänge realisiert, somit sind auch unterschiedliche Spektren und Bestrahlungsstärken realisierbar. Das beschränkt sich zurzeit auf Simulatoren in Zellgröße. Es gibt bereits LED-Sonnensimulatoren für Module am Markt.

1.2. Sonnensimulator Aufbau

Mit der SolarConstant 4000 (siehe Abbildung 1.2) der Firma Atlas kann die Einwirkung der Sonne auf Testobjekte simuliert werden. Dazu ist das Gerät mit zehn Metallhalogenid-Strahlungsquelle

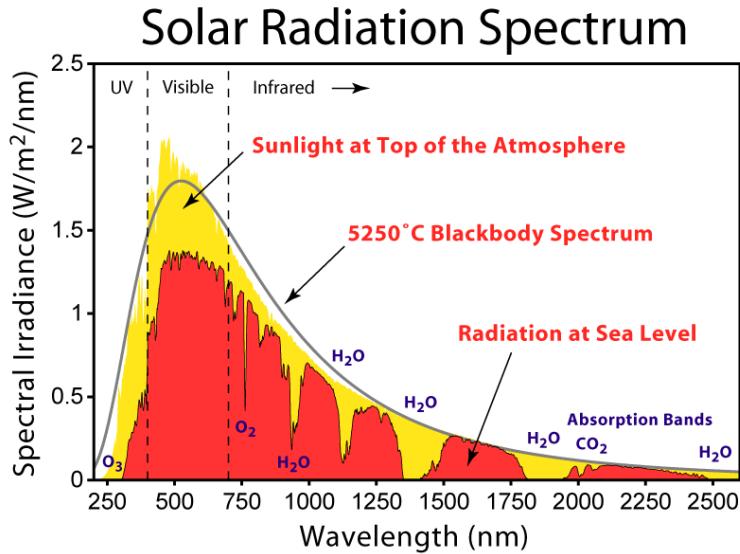


Abbildung 1.1.: Der Vergleich von Spektrum des Sonnenlichtes im Weltall (AM0), der Strahlung eines schwarzen Körpers von 5250°C und dem AM1.5 Spektrum. Quelle: [3]

Lampen zu je 4kW. Durch die Füllung der Lampen mit Halogeniden wird ein kontinuierliches Spektrum erzeugt. Die Leuchten mit Filterscheiben ausgestattet, die das abgestrahlte Licht möglichst dem AM1.5 Spektrum ähnlich macht. Jede Lampe wird von einem eigenen elektronisch geregelten Vorschaltgerät versorgt, welches für einen gleichmäßigen und flimmerfreien Betrieb sorgt. Da die Lampen nicht für eine Heißzündung ausgelegt sind, und dabei eventuell zerstört werden können, ist eine Sperrzeit von 10 Minuten zum Abkülen vorgesehen, bevor die Lampen wieder eingeschaltet werden können. Die Strahlungsleistung lässt sich reduzieren indem das gesamte Lampenfeld in die Höhe gefahren wird. Weiters lässt sich die Bestrahlungsstärke einzelner Lampen über die Variation der elektrischen Leistung des Vorschaltgerätes variieren. Allerdings führt eine Variation der elektrischen Leistung zu einer Änderung der spektralen Strahlungsverteilung. Es wird vom Hersteller empfohlen, die Helligkeit der Lampen nur zwischen 80 und 100 % zu variieren. Die Testobjekte werden mittels Prüfgut-Einschub in einem Windkanal positioniert, dessen obere Abdeckung aus einem geeigneten Solarglas besteht. Die geneigte Glasplatte bewirkt eine Querschnittsreduktion des Luftkanals und somit eine Erhöhung der Strömungsgeschwindigkeit zwischen Zuluft- und Abluftseite. Diese stetige Erhöhung der Geschwindigkeit ist notwendig, damit die sich erwärmende Zuluft eine konstante Kühlwirkung hat, um die Testobjekt auf konstanter Temperatur zu halten.

1.3. Normative Anforderungen

Die Norm IEC 60904-9 [4] stellt Anforderungen an Sonnensimulatoren. Ein Sonnensimulator wird anhand von 3 Kriterien bewertet:

- die spektrale Übereinstimmung mit dem in Tabelle 1.1 aufgelisteten Wellenlängenbereichen. Zum AM1.5 Referenzspektrum sind durch die großen Schranken erhebliche Unterschiede möglich.



Abbildung 1.2.: Der Aufbau des Sonnensimulators: Die 10 Metallhalogenoid Strahler, der Windkanal und die Prüfebene sind zu erkennen

	Wellenlängenbereich in nm	% der totalen Einstrahlung
1	400 - 500	18,4
2	500 - 600	19,9
3	600 - 700	18,4
4	700 - 800	14,9
5	900 - 900	12,5
6	900 - 1100	15,9

Tabelle 1.1.: Spektrale Strahlungsverteilung nach IEC 60904-9

- die Gleichmäßigkeit der Bestrahlungsstärkeverteilung über die Testfläche

$$Non-uniformity(\%) = \left[\frac{\maxirradiance - \minirradiance}{\maxirradiance + \minirradiance} \right] \times (100) \quad (1.2)$$

- die zeitliche Stabilität der Einstrahlung.

$$Temporal(\%) = \left[\frac{\maxirradiance - \minirradiance}{\maxirradiance + \minirradiance} \right] \times (100) \quad (1.3)$$

Tabelle 1.2 gibt die Anforderungen an, nach denen Sonnensimulatoren in den Klassen A,b und C klassifiziert werden. Die Sonnensimulatorklasse ABB bedeutet eine 0,75- bis 1,25-fache Übereinstimmung in allen in Tabelle 1.1 angeführten Schranken, eine Homogenität der Einstrahlung zwischen 2% und 5% im Messbereich, und eine zeitliche Stabilität zwischen 2% und 5%.

Klassifikation	Spektrale Übereinstimmung	Örtliche Homogenität	Kurzzeitstabilität	Langzeitstabilität
A	0,75 - 1,25	2 %	0,5 %	2 %
B	0,6 - 1,4	5 %	2 %	5 %
C	0,4 - 2,0	10 %	10 %	10 %

Tabelle 1.2.: Anforderungen an die 3 verschiedenen Simulatorklassen

1.4. Theorie Referenzzelle

Es gibt verschiedene Solarzelltechnologien. Wirtschaftlich bedeutend sind im Moment nur kristalline Siliziumzellen und Dünnschichtzellen. Die kristallinen Siliziumzellen werden in monokristalline Zellen und multikristalline Zellen unterschieden. Beide zusammen machen über 80% des weltweiten Photovoltaikmarktes aus [5]. Als Referenzzelle für Messungen werden ausschließlich kristalline Zellen verwendet. Eine Referenzzelle ist eine genau vermessene Solarzelle, die als Strahlungsmessgerät dient. Wichtig ist, dass die Referenzzelle eine ähnliche spektrale Empfindlichkeit hat wie die zu messenden Zelle bzw. das zu messende Modul. Siliziumsolarzellen bestehen aus einer großflächigen Diode. Die Sperrsicht ist dabei dem Sonnenlicht ausgesetzt. Gelangt ein Lichtquanten in die Sperrsicht kann aufgrund des inneren Photoeffektes ein Elektron/Loch Paar erzeugt werden. Durch das elektrische Feld in der Sperrsicht werden die Ladungsträger getrennt bevor sie kombinieren können. Elektronen bewegen aufgrund ihrer negativen Ladung entgegen der Feldrichtung in die n-Zone. Löcher wandern in Feldrichtung zur Raumladungsfreien p-Zone. Die Leerlaufspannung einer Solarzelle ist kleiner als die Diffusionsspannung, der Spannung über die Raumladungszone, die der Diffusion von Ladungsträgern entgegenwirkt.

Die unbeleuchtete Solarzelle ist funktioniert wie eine normale Halbleiterdiode, die einen Durchlassstrom von p- nach n-Seite fließen lässt, falls eine Spannung von p nach n anliegt. Bei Beleuchtung wird zusätzlich ein Photostrom erzeugt, welcher proportional zur Bestrahlungsstärke und der Zellfläche ist. Das Eindiodenmodell (siehe Abbildung 1.4) besteht daher aus einer Stromquelle, dazu parallel einer Diode, einen Parallelwiderstand und einem Serienwiderstand. Der Parallelwiderstand fasst Kurzschlüsse zusammen, die realen Solarzellen am Rand oder an den Korngrenzen auftreten können. Mit dem Serienwiderstand werden alle Spannungsabfälle in der Solarzelle erfasst. Eine ideale Solarzelle hat einen Serienwiderstand von null und einen Parallelwiderstand von unendlich.

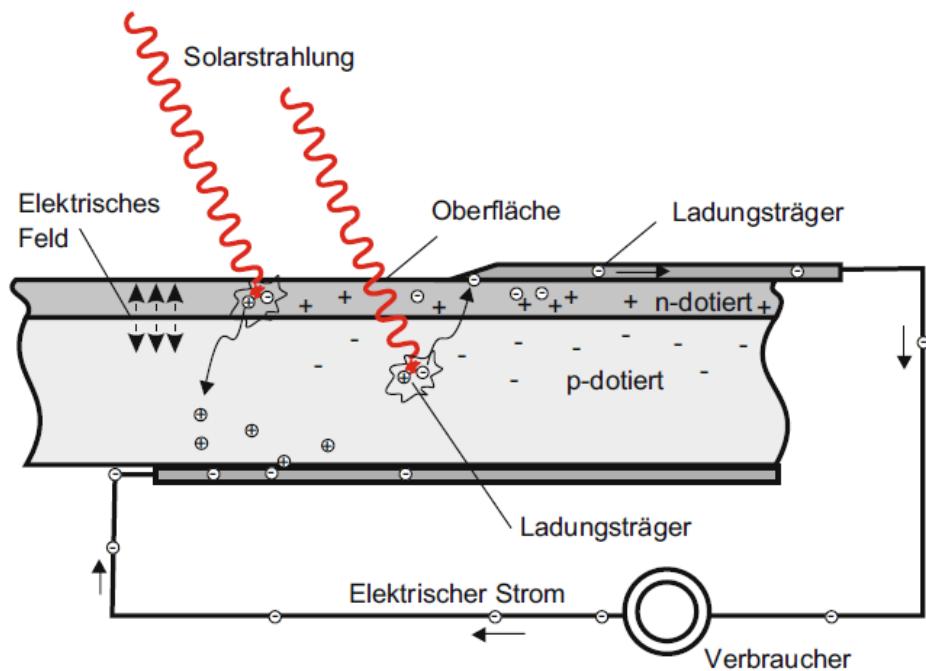


Abbildung 1.3.: Der Schematische Aufbau einer Siliziumsolarzelle. Quelle: [6]

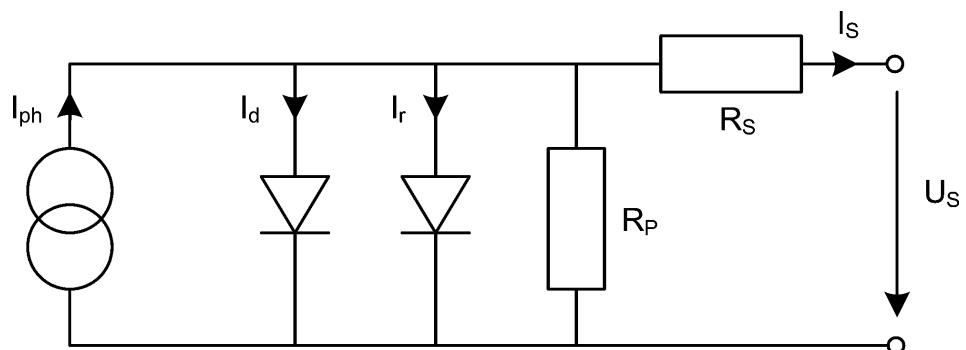


Abbildung 1.4.: Das Zweindiodenmodell einer Solarzelle. Quelle: [7]

Durch den zusätzlichen Photostrom verschiebt (siehe Abbildung 1.5) sich die Diodenkennlinie. Im Leerlauf bzw. Kurzschlussbetrieb gibt die Solarzelle keine Leistung ab. Der Punkt der Kennlinie mit der maximalen Leistung wird als Maximum Power Point (MPP) bezeichnet.

Der Füllfaktor berechnet sich aus Strom im maximalen Leistungspunkt I_{mp} , Spannung im maximalen Leistungspunkt U_{mp} , Kurzschlusstrom I_{sc} und Leerlaufspannung U_{oc} zu:

$$FF = \frac{I_{mp}U_{mp}}{I_{sc}U_{oc}} \quad (1.4)$$

Der Füllfaktor ist das Verhältnis der Fläche des Rechteckes mit den Seitenlängen I_{mp} und U_{mp} zu der Fläche des Rechteckes mit den Seitenlängen I_{sc} und U_{oc} (siehe Abbildung 1.5).

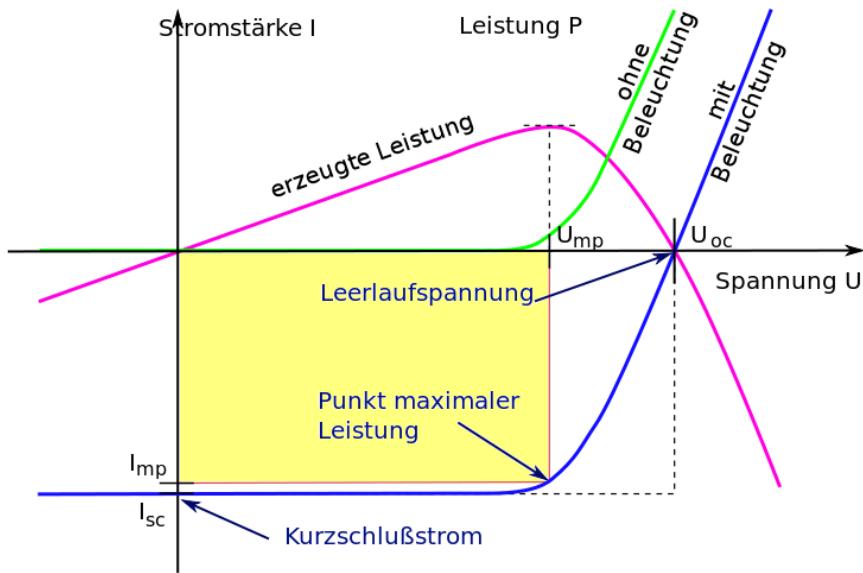


Abbildung 1.5.: Dunkel- und Hellkennlinien

Die Kennlinie der Solarzelle ist von der Einstrahlung abhängig (siehe Abbildung 1.6). Der Kurzschlusstrom ist direkt proportional zur Einstrahlung. Laut IEC 60891 [[8]] berechnet sich die Einstrahlung aus dem gemessenen Kurzschlusstrom, den STC-Kurzschlusstrom, einen relativen Temperaturkoeffizienten und der Zelltemperatur:

$$G = \frac{1000Wm^{-2}I_{RC}}{I_{RC,STC}} [1 - \alpha_{RC}(T_{RC} - 25^{\circ}C)] \quad (1.5)$$

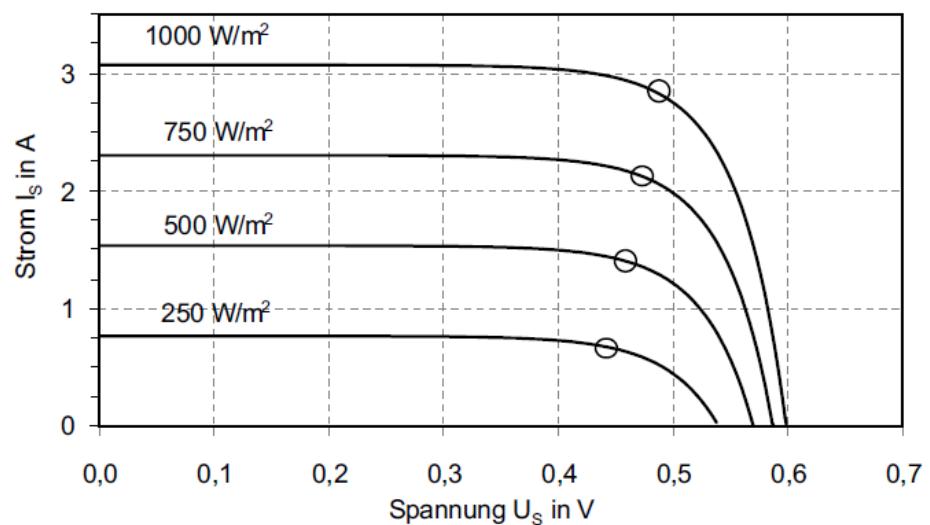


Abbildung 1.6.: Abhangigkeit des MPP von der Einstrahlung

2. Entwicklungsprozess

Diese Kapitel beschreibt die Entwicklung des Roboters und ist in 3 Unterabschnitte gegliedert: das mechanische Design, die Elektronik und die dazugehörige Software. Einige Designelemente waren zu Beginn der Entwicklung vorgegeben. Der Roboter sollte mit Mecanum-Rädern ausgestattet sein. Er sollte eine photovoltaische Messzelle aufnehmen können. Er muss sich im Raum orientieren können. Die Materialien sollten UV beständig sein, da der Roboter in einer Umgebung mit starker UV arbeitet. Die Mecanum-Räder erlauben es dem Roboter sich auch seitwärts fortzubewegen, die Längsachse des Roboters, und somit auch die Messzelle, während des gesamten Messvorganges die Orientierung im Raum beibehält.

2.1. Hardwaredesign und Komponenten

Dieser Abschnitt behandelt die Entwicklung der mechanischen Komponenten, diese sind die 4 Räder und die Chassis des Roboters. Die mögliche Bauhöhe ist durch den Windkanal im Sonnensimulator beschränkt, zusätzlich soll die Messzelle möglichst in der gleichen Höhe wie zu messende Module liegen. Die Größe der Messzelle, zusammen mit dem Raddurchmesser und der Breite der Räder gibt eine Minimalabmessung vor die der Roboter nicht unterschreiten kann. Zusätzlich müssen noch die Motoren, der Akku und die Elektronik im Inneren des Roboters Platz finden. Designprozess:

- Mecanumräder sind festgelegt
- Platz für Messzelle, oben, darunter Elektronik und Motoren
- Mecanum Räder innerhalb des Chassis, die Zelle befindet sich am Dach
- Optische Sensoren zum Verfolgen einer Linie, da auch seitwärts gefahren werden soll, in der Mitte des Roboters an der Unterseite.
- Platz für den Mikrocontroller, die Elektronik, den Akku im Inneren.
- Da wichtige Komponenten wie der Akku später gewechselt wurden, war es gut Reserveplatz zu haben.

Der Roboter ist ein wenig zu groß geworden, die verwendete Messzelle ist kleiner als in den ersten Planungen, dafür ist der Akku größer (und leistungsfähiger) als ursprünglich gedacht. Das Gehäuse war vor den elektronischen Komponenten fertig.

2.1.1. Die Mecanum-Plattform

Mecanum-Räder erlauben einem Fahrzeug, ohne mechanischer Lenkung, sich in jede Richtung zu bewegen. Benannt ist es nach dem schwedischen Unternehmen Macanum, welches dieses Rad 1971 entwickelt hat. Jedes Rad wird über einen eigenen Motor angetrieben, und verfügt über eine separate Ansteuerung. Die Räder bestehen aus einer Felge (siehe Abbildung 2.2), auf der

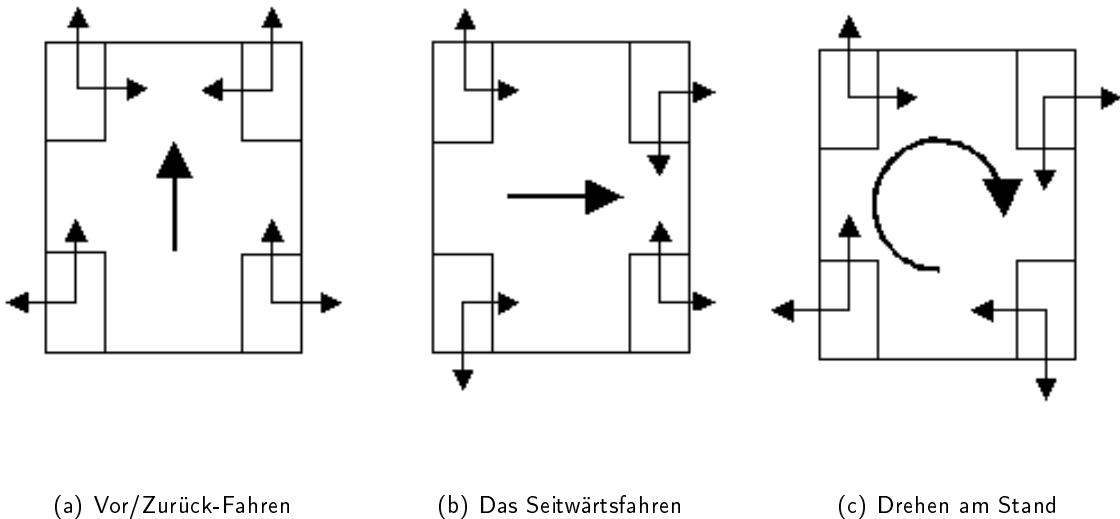


Abbildung 2.1.: Einige mögliche Fahrmanöver eines Mecanum-Rad-Roboters

unter einem Winkel von 45 Grad befestigte, ballige Rollen so angebracht sind, die über den Abrollumfang einen exakten Kreis bilden (siehe Abbildung 2.3). In die 8 Löcher der Felge werden Kugellager eingeklebt. Jedes der vier Räder besteht aus einer Felge, insgesamt 16 Laufrollen, 8 Kugellager, 8 M4x45mm Schraube, 8 M4 Muttern, sowie 16 M4 Beilagscheiben. Das Rad ist mit einer M3 Schraube und einer M3 Mutter an der Motorachse fixiert. Die Felgen und die Laufrollen wurden mit einem 3D-Drucker gefertigt. Die Vorlage namens Mecanum Wheel MK2 stammt von <http://www.thingiverse.com/thing:2473>, und kann unter der Attribution-NonCommercial-ShareAlike 3.0 Unported Lizenz nicht kommerziell verwendet werden. Die Laufrollen so angepasst, dass die Mutter bzw. der Schraubenkopf in der Rolle versenkt ist, damit das Rad insgesamt kompakter ist. Zum bearbeiten wurde OpenSCAD, ein 3D-Compiler, verwendet.

Durch die Schräganordnung der Laufrollen entstehen beim Antreiben eines Rades 2 Kraftkomponenten, eine in Querrichtung und eine in Längsrichtung des Fahrzeuges. Gegeneinander gerichtete Kräfte der einzelnen Räder werden über die Achsen und den Rahmen kompensiert. Die übrigen Kräfte addieren sich zur resultierenden Fahrtrichtung. Auf diese Weise sind durch entsprechendes Ansteuern der einzelnen Räder omnidirektionale Fahrmanöver möglich (siehe Abbildung 2.1). So ist es nicht nur möglich den Roboter in Längsrichtung vor und zurück zu bewegen, sondern auch in Querrichtung (normal zur Längsrichtung) zu bewegen, ebenso sind Drehungen am Stand möglich. Damit braucht ein mit Mecanumrädern ausgestatteter Roboter keine Lenkung. Aber es sind zwar vier Motoren notwendig, damit jedes Rad einzeln angesteuert werden kann.

Mit OpenSCAD entsteht ein dreidimensionales Modell eines Werkstückes. Damit fängt eine 3D-Drucker wenig an. Ein 3D-Druck setzt sich aus sehr vielen Liniensegmenten zusammen. Dazu muss das 3D-Modell in einzelne dünne Schichten und einzelne Linien zerlegt werden. Dieser Vorgang wird als "Slicing" bezeichnet. Dazu wurde das Programm printrface verwendet. Mit OpenSCAD wird zunächst ein stl-File erzeugt, dann produziert printrface daraus die Fahrbefehle für den 3D-Drucker in Form eines g-Files. Als 3D-Drucker wurde ein Ultimaker verwendet. Als Druckmaterial wurde ABS (Acrylnitril-Butadien-Styrol) verwendet, welches in einem Temperaturbereich von 215 bis 230 Grad Celsius per Extruder verarbeitet wird. Die Druckzeit pro Rolle betrug 20 Minuten. Insgesamt mussten 64 Rollen gefertigt werden. Die Druckzeit pro Felge betrug etwa drei Stunden,

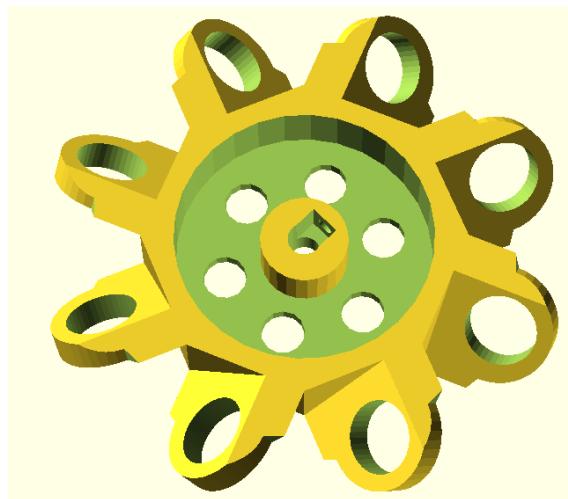


Abbildung 2.2.: Die Felge des Mecanum Rades



Abbildung 2.3.: Ein fertig Rad, montiert am Roboter.

der Roboter benötigte 4 Felgen. Es gibt insgesamt zwei verschiedene Felgentypen, mit nach links oder nach rechts geneigten Rollen. Dazu kommen noch vorbereitende Testdrucke bis die einzelnen Komponenten optimiert waren und der Drucker befriedigend Ergebnisse lieferte. (3D-Druck ist eine schwarze Kunst.) Mehrfach wurde etwa der Durchmesser der Bohrlöcher der Rollen verändert. Nachbearbeitung: Die Löcher für die Kugellager mussten per Hand passend gefeilt werden. Beim Einsetzen der Kugellager brachen manche der Halterungen. Mit Klebstoff konnten die Kugellager auch in die gebrochenen Halterungen eingebaut werden. Die dauerhafte Haltbarkeit dieser Konstruktion ist allerdings mit einem Fragezeichen behaftet.

2.1.2. Chassis

Das Chassis wurde aus Sperrholz gefertigt. Dieses Material wurde gewählt weil es leicht zu bearbeiten, robust und UV-beständig ist. Das Design wurde mit QCAD entwickelt. QCAD ist ein 2-dimensionales CAD Programm. Die Abmessungen des Chassis wurde im Wesentlichen abgestimmt auf:

- Den Durchmesser und der Breite der Mecanum Räder, die innerhalb des Chassis liegen, damit die Räder vor der UV Strahlung geschützt sind, kein Einfluss (Beschattung oder Reflexionen) auf die Messzelle besteht.
- Die Größe der Motoren.
- Den Arduino-Mikrocontroller , die Motorsteuerelektronik und die Messelektronik
- Den Akku zur Stromversorgung, die dazugehörige Akku Spannungsüberwachung
- Die Abmessung der eingekapselten Solarzelle.

Das Chassis besteht aus der Bodenplatte, den Seitenwänden und der oberen Abdeckung mit der Aufnahme für die Messzelle (siehe Abbildung 2.4). Zur Verringerung des Luftwiderstandes sind zwei Seitenwände geneigt (siehe Abbildung 2.5). Die Einzelteile wurden mit einem Lasercutter aus einer Sperrholzplatte geschnitten. Die Einzelteile wurden teilweise miteinander verleimt. Damit eine zukünftige Wartung problemlos möglich ist, wurde der Aufbau verschraubt ausgeführt. Die Motoren sind verschraubt, damit ein Austausch möglich ist. Einzelne Räder als auch die Rollen lassen sich bei Bedarf austauschen. Das Chassis besteht aus der Bodenplatte, den Seitenwänden und der oberen Abdeckung mit der Aufnahme für die Messzelle (siehe Abbildung 2.4). Zur Verringerung des Luftwiderstandes sind zwei Seitenwände geneigt (siehe Abbildung 2.5).

2.2. Steuerungselektronik

Dieser Abschnitt beschreibt die Entwicklung der Elektronik. Die Funktionalität wurde auf verschiedene Module aufgeteilt. Das erleichtert die Platzierung der Elektronik innerhalb des Messroboters, und den Austausch von Komponenten im Fehlerfall. Das Gehirn des Roboters ist ein Arduino Mega 2560 Mikrocontroller Board. Darauf aufgesteckt ist ein sogenanntes Shield, eine selbst entwickelte Platine, das die $\pm 5V$ Spannungsversorgung, die Schnittstellen zu den anderen Platinen und einen SD-Karten Einschub beheimatet. Es gibt 3 Messplatinen, eine für den Kurzschlussstrom der Messzelle, zwei für Temperaturmessungen. Eine Platine überwacht den Ladezustand des Akkus. Die Steuerung der vier Motoren ist auf einer Platine zusammengefasst. Die einzelnen Komponenten sind wie in Abbildung 2.6 auf der Bodenplatte angeordnet.

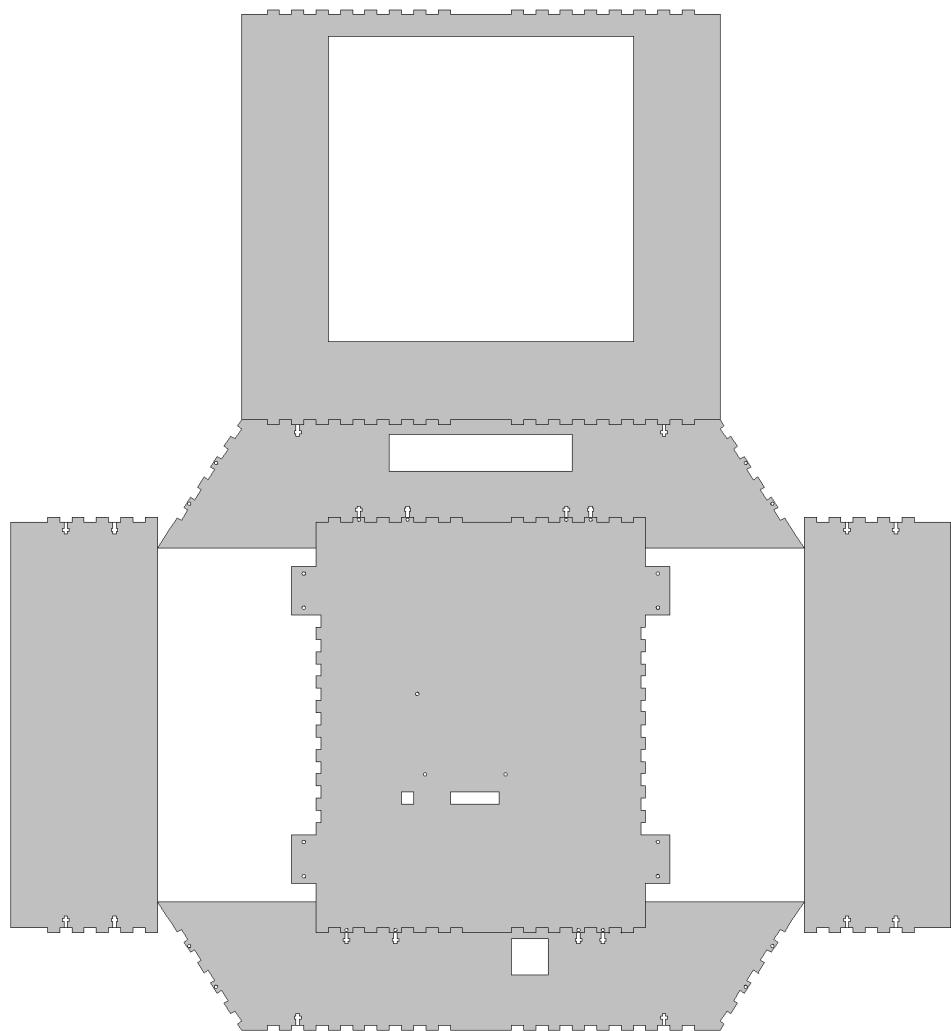


Abbildung 2.4.: Die Schnittvorlage des Chassis

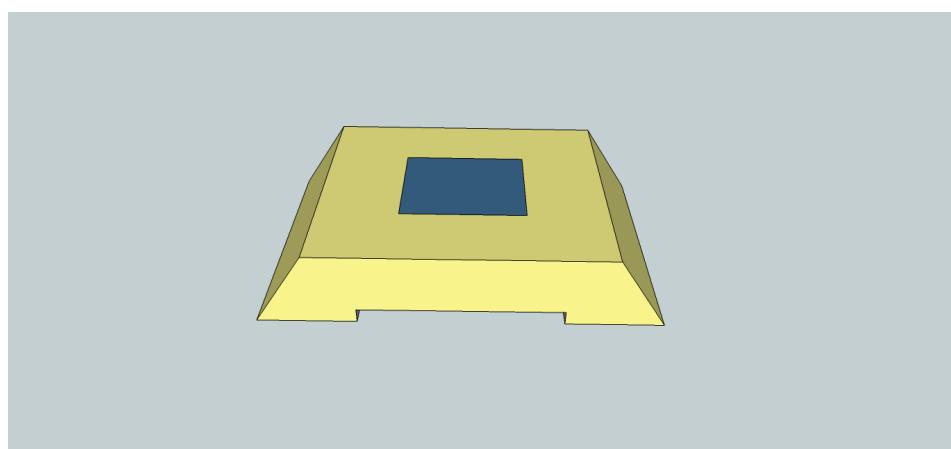


Abbildung 2.5.: Ein 3D-Modell der Chassis

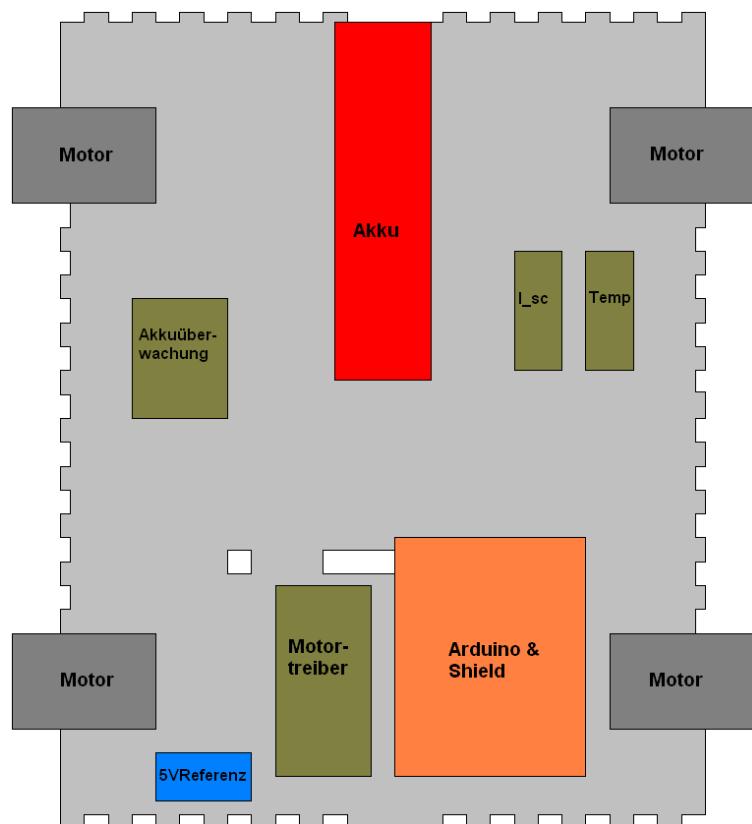


Abbildung 2.6.: Die Anordnung der Elektronik auf der Bodenplatte im Roboter

2.2.1. Motorensteuerung

Die Motorplatine 2.7 besteht aus zwei integrierten H-Brücken-Bausteinen (IC1, IC2), einer 16 poligen Buchse und fünf zweipoligen Buchsen für die Stromversorgung und den Anschluss der vier Motoren. Bei den integrierten H-Brücken-Baustein handelt es sich um einen NJM2670 [9], welcher zwei H-Brücken in einem Baustein vereint, zusätzlich verfügen diese Bausteine über eine interne Logik, welche Kurzschlüsse über die Brücke unmöglich macht. Die Drehzahl der Motoren wird mit Pulsweitenmodulation variiert. Dabei ist es wichtig, dass die Transistoren in der Brücke schnell genug schalten können. Die PWM-Ausgänge des Arduino arbeiten mit einer Frequenz von etwa 500Hz [10]. Die Gleichspannungsmotoren sind Getriebemotoren mit der Bezeichnung RB 35, und einem Übersetzungsverhältnis von 1:200. Die maximale Versorgungsspannung wurde mit 12V angegeben. Die Leerlaufdrehzahl beträgt bei 12V Versorgungsspannung 20 Umdrehungen pro Minute. Der Roboter wird mit einem Lithium-Ionen-Akku versorgt. Die Spannung beträgt, abhängig vom Ladezustand zwischen 16,4V und 15,0V. Damit es zu keiner Überlastung der 12V-Motoren, und die Geschwindigkeit des Roboters nicht zu groß wird, kommt wurde bei der Pulsweitenmodulation ein maximales Tastverhältnis von 50% eingehalten.

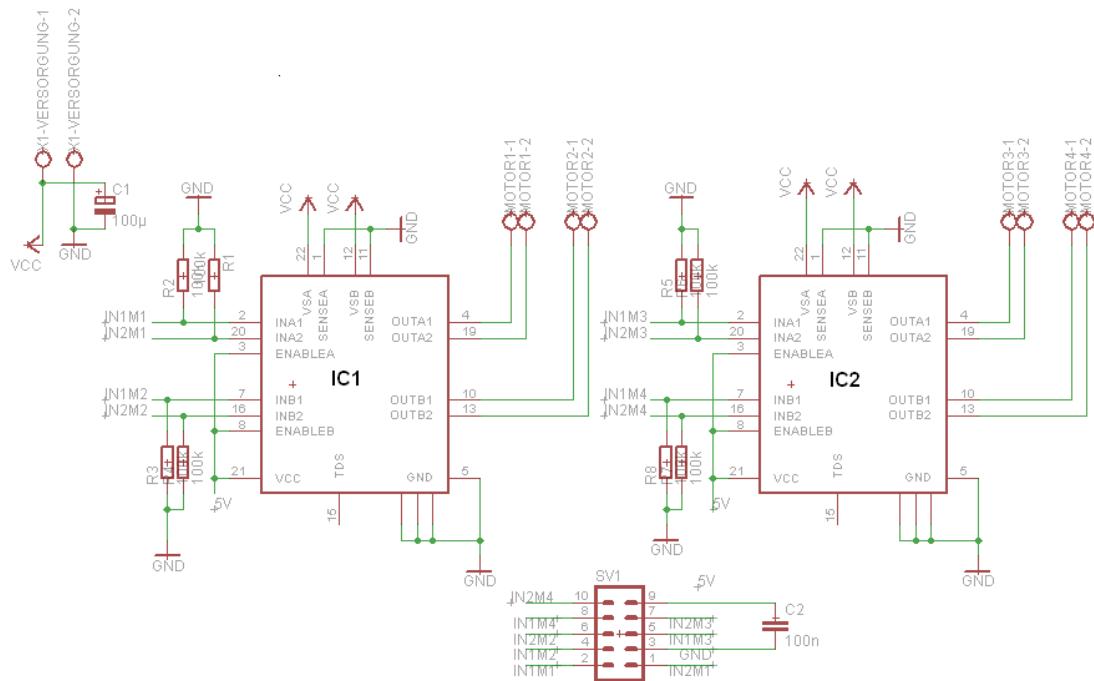


Abbildung 2.7.: Der Schaltplan der Motortreiberplatine.

2.2.2. Optische Sensorik

Die optische Sensorik dient der Positionsbestimmung. Der Roboter bewegt während der Messung auf Holzplatten, die aus Gründen, welche unten genauer erläutert werden, schwarz sind, der Roboter orientiert sich an einer aus geraden Teilstücken bestehenden weißen Linienfolge. Als Sensor wurde ein CNY70 verwendet. Der CNY 70 ist ein Reflexionssensor, bestehend aus einer Infrarot. Der Sensor wird in einem fixen Abstand zu einer Ebene montiert. Eine LED sendet Licht mit

950nm aus, welches an der Ebene reflektiert wird. Damit können Unterschiede des Reflektionkoeffizienten gemessen werden. Deshalb sind die Markierungen weiß auf schwarzen Untergrund. Eine Anordnung von neun solcher Sensoren in einem 3-mal-3 Array kann sowohl vertikale als auch horizontale Linien sowie Ecken erkennen. Zusätzlich gibt es einen 10. Sensor, der die Stopmarkierungen für die Messungen erkennt. Die neun Sensoren sind im Rastermaß 15,2 mm (= 0,6 Zoll) angeordnet. Der Stopsensor hat einen Abstand von 40,6mm zu dem Sensorarray. Der Strom durch die LEDs ist mittels Potentiometer einstellbar. Die Platine ist zweiseitig ausgeführt, allerdings sind alle Bauteile an der Unterseite der Platine montiert, bis auf die OPVs in SMD-Ausführung. Das ursprüngliche Layout wurde um folgende zwei Änderungen modifiziert: - Auf die Steuerleitung des Transistors wurde vergessen. Da es noch freie Pins am 16-poligen Stecker gab, wurde diese Verbindung mittels Draht hergestellt. - Die Platine wird mit der 5 Volt Versorgung der restlichen Elektronik versorgt, welche konstanter als die 7805-Variante ist, und um Probleme mit zwei verschiedenen 5 Volt Versorgungen zu vermeiden. Dazu wurde der ursprünglich vorgesehene 7805 ausgebaut, und eine weiterer freier Pin des 16-poligen Steckers verwendet. Die LEDs sind parallel

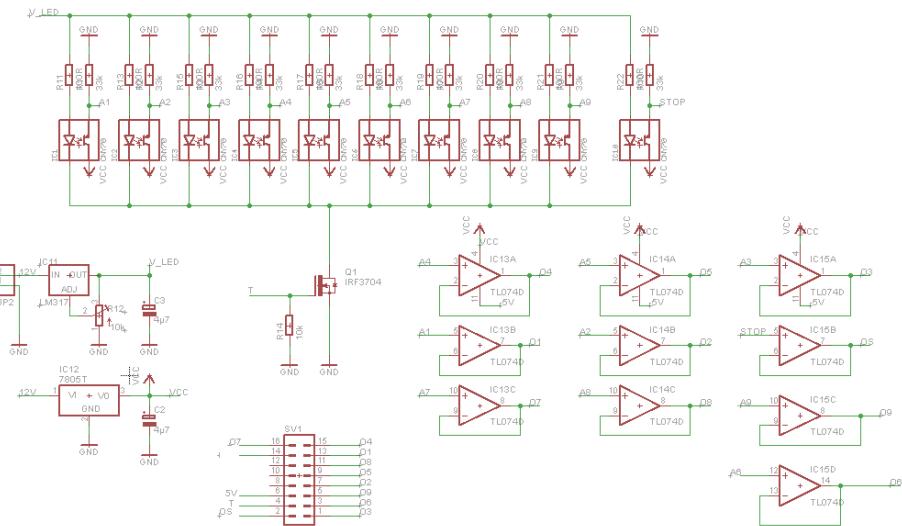


Abbildung 2.8.: Der Schaltplan des Sensorarrays.

geschaltet und werde alle über den Feldeffekttransistor Q1 des Typs IRF530 geschaltet. Der Strom durch die LEDs wird mit einem LM317 geregelt. Da der Widerstand als Potentiometer ausgeführt ist lässt sich der Strom einstellen. Trotz einer nicht konstanten Versorgungsspannung kann so einer konstanter Strom geliefert werden. Um das analoge Messsignal der Phototransistoren nicht zu belasten, werden die analogen Photosignale über einen Spannungsfolger geführt. Die OPVs benötigen allerdings eine symmetrische Versorgung mit ± 5 V. Beide Spannungen werden über den Stecker geliefert. Es gibt weiteren Stecker mit 2 Pins, welche die Spannungsversorgung mit 16 Volt und GND sicherstellt. Die Platine verfügt um drei Bohrlöcher mit jeweils drei Millimeter Durchmesser zu Befestigung. Anschlüsse: Es ist eine 16-polige Leiterplattenbuchse für die Analogsignale, 5V-Versorgung, und des Digitalsignal zum Schalten des FET. Weiters ist ein zweipoliger Stiftstecker für die Stromversorgung vorhanden.

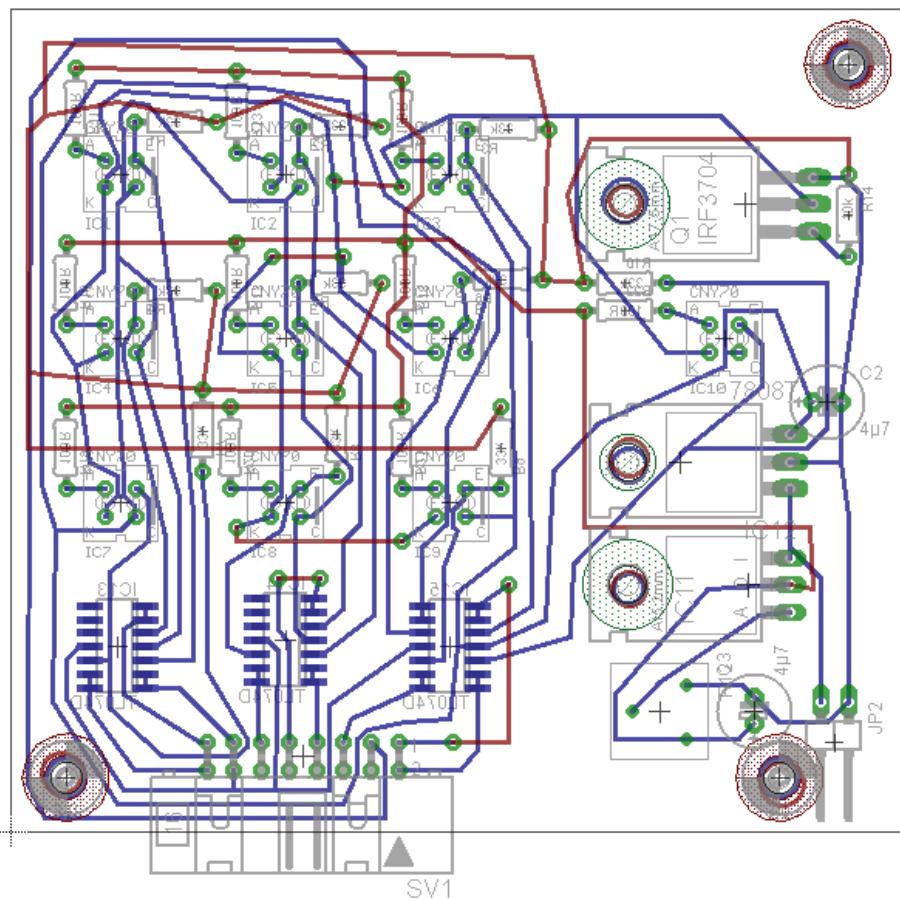


Abbildung 2.9.: Das Layout des Sensorarrays. Die optischen Sensoren sind gelb markiert.

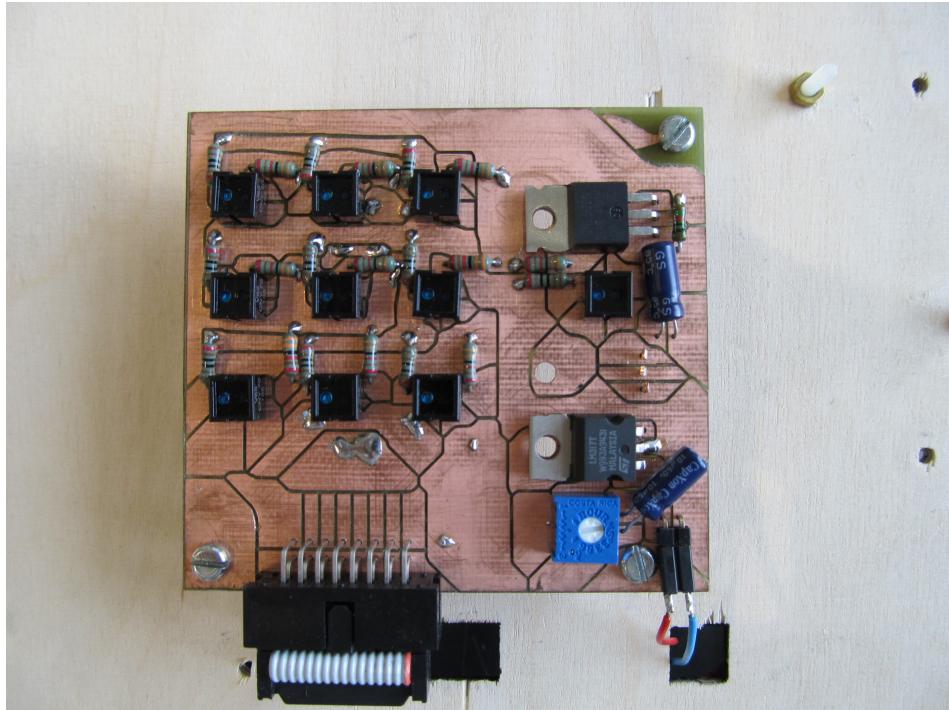


Abbildung 2.10.: Die fertige Platine des Senorarrays.

2.2.3. Spannungversorgung

Die Energie zum Betrieb des Roboters stammt aus einem Kokam 14,6V 4000mAh Lithium-Polymer-Akkumulator. Es wird empfohlen vor jeden Messzyklus den Akku zu laden. Ein voller Akku reicht für mehrere Stunden Dauerbetrieb. Der Akku ist mit einer im Anschlusskabel integrierten Schmelzsicherung geschützt. Zum Laden des Akkus muss der Akku vom Roboter abgesteckt werden und an das Ladegerät angeschlossen werden. Zum Schutz vor Kurzschlägen ist der Akku mit einer im Anschlusskabel integrierten Schmelzsicherung gesichert. Da zum Laden der Akku aus dem Roboter entfernt und an das Akkuladegerät angeschlossen werden muss, könnte versehentlich der Akku kurzgeschlossen werden. Zum Laden wird ein Ultramat 16 S verwendet, der über einen Balanceranschluss verfügt, und somit alle vier Zellen des Akkus auf die gleiche Spannung lädt. Der Akku kann mit bis zu 200A belastet werden, allerdings liegt der maximale Strom aller 4 Motoren und der restlichen Elektronik bei einem Ampere. Die Motoren werden direkt mit der Akkuspannung versorgt. Die Versorgungsspannung schwankt zwischen der vollen Akkuspannung 16,4V und 15V nach mehr als 10 Messfahrten. Die 5 Volt Versorgung des Mikrocontrollers wurde mit einem RECOM R-785.0-1.0 DC-DC Konverter gelöst. Er ist pinkompatibel zu einem 7805, allerdings mit einer größeren Dicke, und bietet die Vorteile einer geringen Verlustleistung und einer sehr genauen Ausgangsspannung. Der Ladezustand der einzelnen Zellen des Akkus wird mit einer eigens dafür entwickelten Schaltung überwacht. Sollte die Spannung einer Zelle unter 3,3V sinken, wird die Stromversorgung zum Roboter mittels FET unterbrochen, und ein akustisches Warnung ausgegeben. Damit wird eine Tiefentladung verhindert, was zu einer irreversiblen Schädigung und einem Kapazitätsverlust führen kann. Eigentlich könnte der Akku bis zu 2,7V pro Zelle entladen werden (siehe Abbildung ??). Die Analog-Digitalwandlung des Arduino benötigt eine Referenzspannung, entweder die Versorgungsspannung, was jedoch zu ungenau ist. Als Spannungsreferenz wird

daher ein LT1021 5V Präzisionsspannungsquelle verwendet. Da dieses Bauelement nachträglich hinzugefügt wurde, ist es auf einer eigenen kleinen Platine untergebracht.

2.2.4. Mikrokontroller

Der Arduino Mega 2560 ist ein Mikrocontroller-Board. Es ist eine Open-Source Elektronikentwicklungsplattform und richtet sich an Elektronikneulinge wie auch an Profis. Den Mikrocontroller des Arduino-Boards bildet ein ATmega2560 der Firma Atmel. Dabei handelt es sich um einen 8-Bit-Mikrocontroller. Er läuft mit 16 MHz Taktfrequenz und besitzt 256 KByte Flash, wovon 8 KByte für den bootloader verwendet werden, für den Programmcode, 8 KByte SRAM, sowie 4 KByte EEPROM. Es verfügt über 54 digitale I/O Pins, davon können 15 als PWM-Ausgänge verwendet werden. Weiters gibt es 16 analoge Eingänge die über einen 10-Bit Analog-Digital-Wandler verfügen. Eine USB Schnittstelle ist zur Programmierung und Kommunikation vorhanden. Der Mikrocontroller arbeitet mit einer Betriebsspannung von 5V. Grundsätzlich kann die Stromversorgung über die USB Schnittstelle erfolgen. Für einen beweglichen Roboter ist diese Option nicht praktikabel. Es besteht auch die Möglichkeit über eine Buchse eine Versorgungsspannung von 7 bis 12 Volt anzulegen, der Akku liefert aber eine höhere Spannung. Aus der Akku-Spannung, wird eine stabile 5 Volt Spannung gewonnen. Diese 5 Volt werden direkt mit dem 5 Volt Pin des Arduino-Boards verbunden. Die 5-Volt Spannungsversorgung befindet sich auf dem Shield (der Aufsteckplatine). Dabei handelt es sich um einen kompakten DC-DC Konverter.

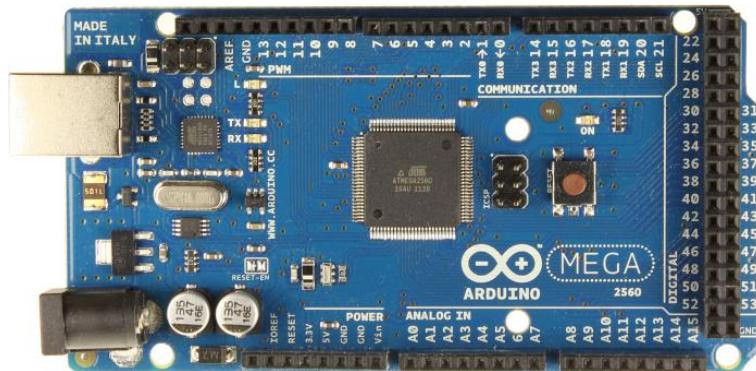


Abbildung 2.11.: Arduino Mega 2560

2.2.5. Temperatursensoren

Der Spannungsabfall an einem temperaturstabilen 100Ω Widerstand wird mit dem Spannungsabfall an einem PT100 verglichen. Als Stromquelle dient eine sehr präzise REF200-Stromquelle[11], welche zwei mal 100 Mikroampere $\pm 0.5\%$ liefert. Abbildung 2.12 zeigt den Schaltplan, Abbildung 2.13 zeigt das Platinenlayout.

2.3. Softwareentwicklung

Dieser Abschnitt beschreibt die Entwicklung des Softwareteils. Es wird kurz auf die Entwicklungs-Umgebung eingegangen. Dann folgt die Auswertung der optischen Sensoren. Schließlich wird die Programmstruktur erklärt. Der gesamte Code ist im Anhang gelistet.

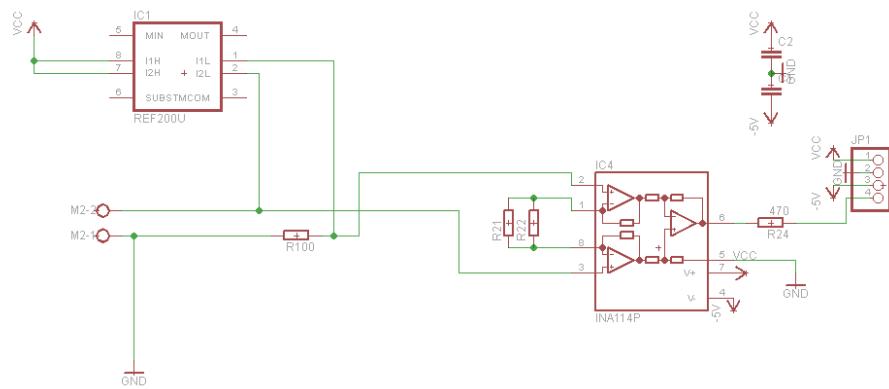


Abbildung 2.12.: Schaltung der Temperaturmessung

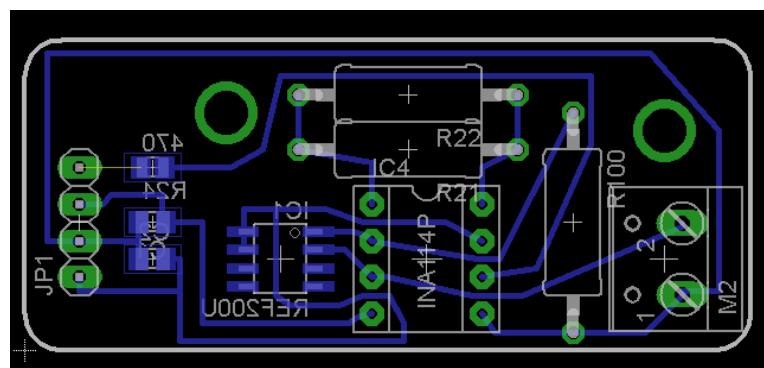
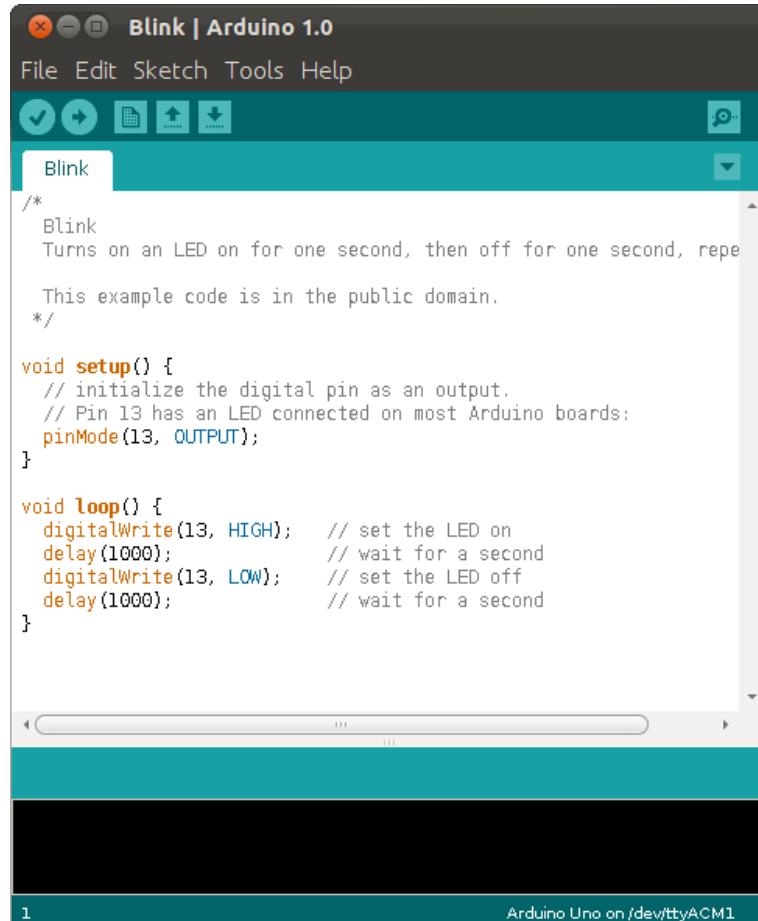


Abbildung 2.13.: Layout der Temperaturmessung

2.3.1. Entwicklungsumgebung

Der Mikrocontroller auf den Arduino Board wird mit der Arduino Programmiersprache, welche auf Wiring basiert und sehr C-artig ist, in der Arduino Entwicklungsumgebung (siehe Abbildung 2.14, welche auf Processing basiert, programmiert. Die Entwicklungsumgebung ist auf das notwendigste beschränkt, und erlaubt einen raschen Einstieg. Die umfassenden Beispielprogramme helfen beim Einarbeiten. Darauf hinaus gibt es eine große und aktive Community. Ein Arduino Programm besteht aus zwei Teilen: Einer Setup-Routine die einmal zu Programmstart durchlaufen wird und einer Endlosschleife. In der Schleife wird das eigentlich Programm abgearbeitet.



The screenshot shows the Arduino IDE interface with the title bar "Blink | Arduino 1.0". The menu bar includes File, Edit, Sketch, Tools, and Help. Below the menu is a toolbar with icons for Save, Run, Upload, and others. The main window displays the "Blink" sketch code. The code is as follows:

```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repe
  This example code is in the public domain.
*/

void setup() {
  // initialize the digital pin as an output.
  // Pin 13 has an LED connected on most Arduino boards:
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);      // set the LED on
  delay(1000);                // wait for a second
  digitalWrite(13, LOW);       // set the LED off
  delay(1000);                // wait for a second
}
```

The status bar at the bottom indicates "1" and "Arduino Uno on /dev/ttyACM1".

Abbildung 2.14.: Arduino Entwicklungsumgebung

2.3.2. Auswertung der optischen Sensoren

Um den Einfluss des Streulichtes der starken Einstrahlung im Sonnensimulator auf die Optoreflektoren CNY70 zu vermeiden, wurde eine differentielle Messmethode angewandt. Die Messwerte bei ausgeschalteter Infrarot-Leuchtdioden werden von den Messwerten mit eingeschalteter Leuchtdioden abgezogen. Das wird für alle neun Optoreflektoren des Sensorarray, sowie den Stoppsensor, gemacht. Mit den korrigierten Werten der einzelnen optischen Sensoren wird die Lage der Linie relativ zum Mittelpunktes des Arrays bestimmt.

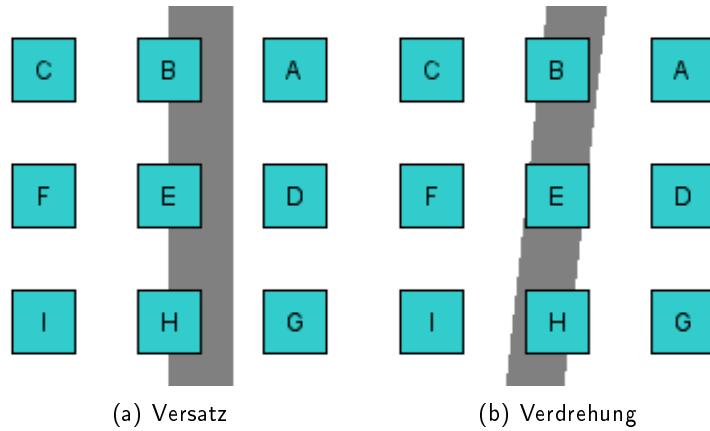


Abbildung 2.15.: Die mögliche Abweichungen der Linie von der Idealposition

Die Höhe des Messwertes der optischen Sensoren ist abhängig von der Helligkeit des Untergrundes. Unter der Annahme dass sich eine helle gerade Linie unter dem Sensorarray befindet funktioniert folgende Formel:

$$x_1 = \frac{C - A}{2A - 4B + 2C}$$

$$x_2 = \frac{F - D}{2D - 4E + 2F}$$

$$x_3 = \frac{I - G}{2G - 4H + 2I}$$

$$d = \frac{x_1 + x_2 + x_3}{3}$$

$$k = \frac{x_1 - x_3}{2}$$

2.3.3. Auswertung ADCs

Aufgrund von Rauschen der ADC-Werte, wurden alle Messwerte über 500 Einzelmessungen gemittelt. Die Quelle des Rauschens wurde nicht ermittelt. Der notwendige Messzeit dafür beträgt unter einer halben Sekunde. Die optischen Sensoren werden ebenfalls mit ADCs ausgewertet, die Behandlung erfolge im vorigen Unterabschnitt.

2.3.4. Programmablauf

Das Programm startet im Modus Start, der Roboter steht, und wartet 60 Sekunden, eine Pufferzeit damit nach dem Einschalten des Roboters die Lade des Sonnensimulator hineingeschoben und die Klappe geschlossen werden kann. Nach Ablauf dieser 60 Sekunden wechselt das Programm in den

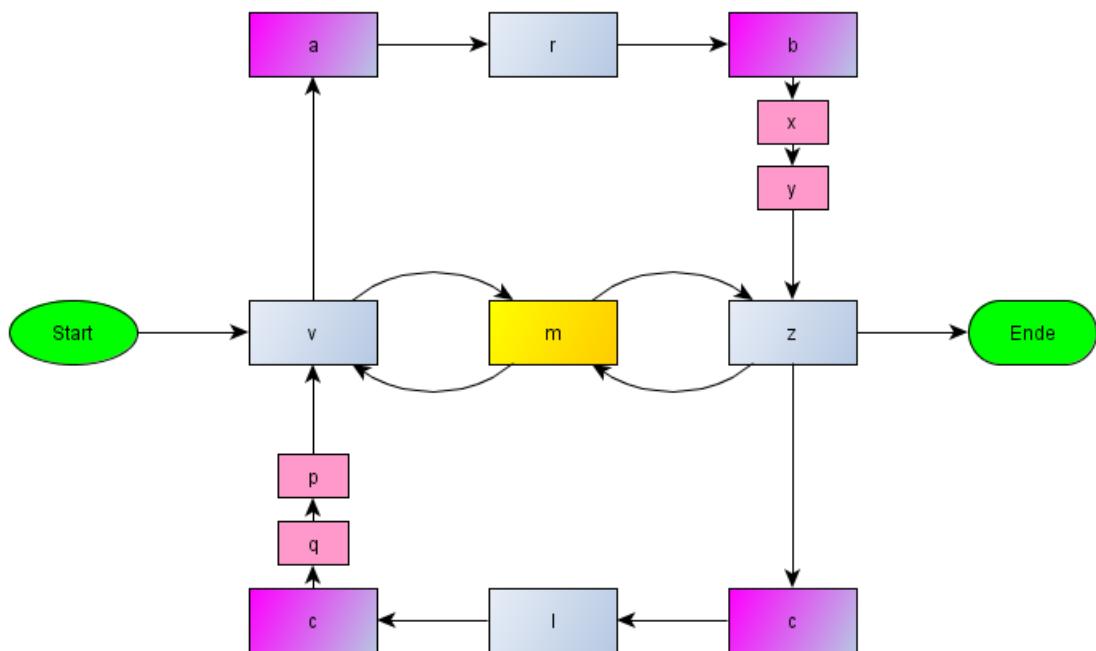


Abbildung 2.16.: Programmablauf

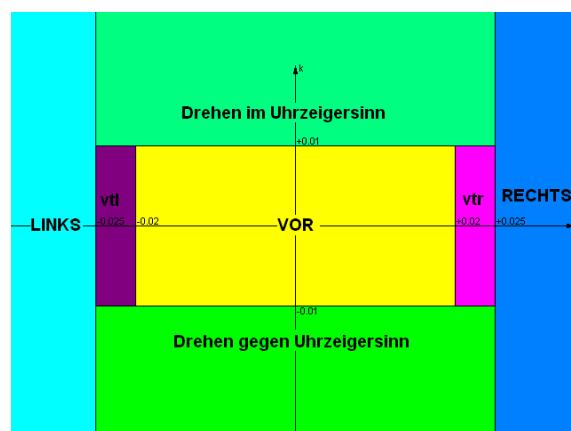


Abbildung 2.17.: Schranken der Regelung

Modus Vorwärts. In diesem Modus fährt der Roboter in Vorwärtsrichtung. Abweichungen von der Idealspur werden erkannt und korrigiert. Wird mit dem Stopsensor eine Stopmarkierung erkannt, wechselt das Programm in den Messmodus. Ein Arduino Programm besteht grundsätzlich aus zwei Teilen: Eine Setup Routine, die einmal durchlaufen wird, und diverse Pins als Eingänge bzw. Ausgänge festlegt. Eine so genannte Loop, die immer wieder durchlaufen wird, eine in der Loop integrierte Zeitverzögerung ist sinnvoll.

Das Einschalten des Roboters ist der Start des Programms. Nach dem Start gibt es eine Verzögerung von 60 Sekunden, die dazu dient, die Einschublade des Sonnensimulators zu schließen. Es wird davon ausgegangen, dass der Roboter auf der Messbahn steht, das heißt das Sensorarray befindet sich über der Messbahn. Die gesamte Bewegungssteuerung ist als Zustandsautomat ausgeführt. (Ablauf2.png Bild der möglichen Zustände). Mode s (Start): Zeitverzögerung von 60s, Übergang zu Mode v (Vorwärtsfahrt) Mode z (Rückwärtsfahrt): Regelung zur Spurhaltung, Übergang zu Mode m (messen) bei erkannten Messpunkt, bei erkannter Ecke Übergang zu Mode a (Fahrt um die Ecke) Mode a, Mode c: Eine um die Ecke Fahrt, bestehend aus zwei Einzelbewegungen: einer Geradeausfahrt bis die obere Sensorreihe (A, B und C) dunkel detektieren, gefolgt von einer Seitwärtsfahrt mit 700ms Dauer. (Funktionierende Lösung! Daher wurde keine Variante ohne Zeitverzögerung gesucht.) Mode b, d: eine Fahrt der Dauer 850ms, in Richtung vor bzw. zurück, Übergang in den Modus der Positionskorrektur (x & y bzw q & p). Messmodus siehe unten. (Screenshot der Messdatei.) Beendigung des Messmodus (Zustand m, Messen): Nach dem Speichern geht der Zustandsautomat wieder in den vorigen Bewegungsmodus über, dazu gibt es eine Hilfsvariable.

Ende der Messfahrt: Im Zustand z (und nur im Zustand z) wird das Ende der Linie als Übergangsbedingung in den Zustand e (=Ende) gewertet.

Der Roboter beginnt nach Ablauf der Einschaltverzögerung seine Vorwärtsfahrt. (Bild mit den Bewegungsrichtungen des Roboters: vorwärts, rückwärts, links und rechts!) Die Geradeausfahrt kann unterbrochen durch einen Messpunkte unterbrochen werden. Wenn der Messpunktsensor einen Wert höher als eine bestimmte Schaltschwelle erhält wird in den Messmodus gewechselt. Messmodus: Zeitverzögerung um 250 ms, zur Vermeidung von elektromagnetischen Störungen durch die Motorströme. Messschleife von 500 Einzelmessungen des Zellenkurzschlussstromes, der Zelltemperatur und der Umgebungstemperatur. Die Einzelmesswerte werden zuerst summiert und nach der Schleife durch 500 dividiert. Schreiben auf SD Karte: Zeitpunkt in ms, zu dem die Messung gestartet hat, dann die 3 ADC Werte, Messpunkt, Messreihe und Messspalte. (Falls ein Messwert verloren geht, was vorkommen kann, wenn ein Messpunkt übersehen wird, kann die Position ermittelt und die fehlenden Messwerte geschätzt werden).

Die Messstrecke setzt sich aus geraden Linienelementen zusammen, die durch 90° Ecken verbunden sind. Bei der Hälfte der Ecken ist die Linie nicht durchgehend, sondern als mit einem 15mm breiten Abstand ausgeführt. Das war nicht geplant, hat sich aber in der als funktionierende Lösung erwiesen. Das Seitwärtsfahren des Roboters funktioniert, aber es sind größere Abweichungen im Versatz und der Verdrehung möglich. Daher kann der Roboter in einer ziemlich ungünstigen Position an der Ecke stehen. Der Algorithmus des um die Ecke Fahrens besteht aus einer Fahrt (ohne Regelung!) in die bisherige Bewegungsrichtung solange bestimmte Sensoren des Arrays nicht dunkel detektieren, gefolgt von einer Bewegung von 90° zur bisherigen Bewegungsrichtung ohne Regelung für eine definierte Zeit. Diese beiden Bewegungen können im schlimmsten Fall die ungünstige Position des Roboters so weit verschlimmern, dass eine Weiterfahrt nach der Ecke unmöglich ist. Durch die Modifikation der Ecke, in ein Ende der Linie mit einem davon seitwärts versetzten Anfangs, vereinfacht sich der prinzipielle Algorithmus. Bis zum Ende der Linie ist es der normale (inkl. Regelung) Fahralgorithmus, danach wird 90° zur bisherigen Bewegungsrich-

tung, d.h. vor oder zurück, mit einer definierten Zeit zur Überbrückung der Lücke ohne Regelung gefahren. Dann steht der Roboter auf der Linie, aber eventuell mit Versatz oder einer Verdrehung.

Regelung bei der Vorwärts bzw. Rückwärtsfahrt. A bei der Vorwärtsfahrt schwarz, G bei der Rückwärtsfahrt schwarz. Damit die Ecken keinen Einfluss (= ungewollte Verdrehung, wäre sehr schlecht als Ausgangslage für das um die Ecke fahren). Da es lange ohne Probleme funktioniert hat wird kein vermeidlich sinnloser Programmteil gelöscht. Die Auswirkungen sind nur durch lange Testfahrten zu ermitteln. In der Regelung für das Fahren in Richtung "vor" gibt es neben der Bewegung für das Vorwärtsfahren noch 6 andere Bewegungsarten. Drehen am Stand mit und gegen den Uhrzeigersinn, links und rechtsfahren, und dann noch zwei Bewegungen die sich aus gleichzeitig drehen und vorwärts fahren zusammensetzen. Die Art der Bewegung ist von zwei Variablen dem seitlichen Versatz d und der Verdrehung k . Liegen beide Variablen innerhalb enger Grenzen fährt der Roboter geradeaus vor. Für das Rückwärtsfahren ist die Regelung analog.

Unabhängig von der Regelung des Seitwärtsfahrens gibt es mechanische Kontaktlemente die eine zu starke Verdrehung des Roboters beim Seitwärtsfahren gar nicht zulassen. Diese Kontaktlemente bestehen aus leichtem Schaumstoff und sind auf die Vorder- bzw. Rückseite des Roboters aufgeklebt.

Die Ecken müssen als solche erkannt werden, bevor sie die normale Fahrregelung stören. Wird eine Ecke erkannt wird der normale Fahrmodus unterbrochen, und in einem speziellen Eckenmodus gewechselt. Es gibt zwei Arten von Ecken: - einmal die klassische Ecke beim Wechsel vom Vor- bzw. Rückwärtsfahren nach rechts. - Die nach Rechts-Fahren Bahn endet, anstatt in einer Ecke in die Vor- bzw. Rückwärtsbahn zu enden. Damit ist auch das um die Ecke fahren auch unterschiedlich. Im Vor- bzw. Rückwärts-Fahren wird die Fahrt ohne Regelung solange fortgesetzt bis die obere bzw. untere Zeile der Sensormatrix schwarz sieht, dann fährt der Roboter ohne Regelung einige hundert Millisekunden nach rechts. Erst dann wird in den nach Rechts-Fahr-Modus gewechselt. Für den Rechts-fahr Modus endet die Linie abrupt. Der Roboter fährt einige hundert Millisekunden vor bzw. zurück. Dann steht er auf der Führungslinie. Bevor in den eigentlichen Vor- bzw. Rückwärtsfahrmodus gewechselt wird, wird der Roboter zur Linie hin ausgerichtet. Die Unterschiedliche Behandlung der Ecken ist notwendig, weil im Rechtsfahren größere Abweichungen von der Ideallinie auftreten als im Vor- und Zurückfahren. Folge der Linie: Der Roboter fährt standardmäßig gerade aus. Wird ein Versatz zur Linie erkannt, wird dieser Versatz durch eine Bewegung normal zur Linie minimiert. Eine Verdrehung zur Linie wird durch Verdrehen des Roboters korrigiert. Ein minimaler Versatz, bzw. eine minimale Verdrehung ist dabei zulässig. Das vorwärts- und rückwärts-Fahren funktioniert dabei besser als das nach rechts-Fahren. Mecanum-Räder haben eine Vorzugsrichtung. Haltepunkte: Haltepunkte werden mit dem Haltepunktsensor erkannt, wenn dieser den Schwellwert für weiß überschreitet. Damit ein Haltepunkt nicht mehrfach erkannt werden kann, gibt es eine Verriegelung in der Software, die einen neuen Haltepunkt erst wieder zulässt wenn der Haltepunktsensor in der zwischenzeit schwarz gesehen hat.

3. Kalibration

Dieses Kapitel beschreibt die Kalibrierung der Messsensoren. Es gibt drei Messplatinen, eine für die Messung des Kurzschlussstromes der Messzelle, und zwei identisch aufgebaute Messplatinen zur Messung der Zelltemperatur beziehungsweise der Umgebungstemperatur. Die Strom-Messplatine wandelt den Kurzschlussstrom der Messzelle in eine Spannung um, die in einem für den ADC des Arduino auswertbaren Bereich (0 bis 5 Volt) ist. Die beiden Messschaltungen zur Temperatormessung wandeln die Größe der temperaturabhängigen Messwiderstände, jeweils ein Pt-100, in eine Spannung um. Die ADC-Messwerte wurden über die USB-Schnittstelle des Roboters ausgelesen. Dafür benötigt der Roboter ein einfaches Kalibrierprogramm.

3.1. Temperatursensoren

Die Temperatur der Messzelle wird mit einem von unten aufgeklebten flächigen Folienmesswiderstand gemessen. Ein zweiter Pt100, in kompakter Dünnschichtbauweise ausgeführt, misst die Temperatur der Umgebung. Beide Messkreise sind identisch aufgebaut, dennoch führen Bauteiltoleranzen zu leicht unterschiedlichen Verstärkungen. Für die Kalibration der Temperaturmesselektronik wurden die Pt-100 Widerstände durch einen hoch genauen, einstellbares und kalibriertes Messnormal ersetzt. Der Widerstand wurde im Bereich von 100 bis 122 Ω in 1 Ω Schritten verändert, was einer Temperatur von 0 bis 55 °C entspricht. Die ADC-Werte wurden über die USB-Schnittstelle des Arduino ausgelesen. Es zeigte sich, dass die beiden Schaltungen leicht unterschiedliche Verstärkungen haben. Daher mussten jede Temperatormessschaltung separat kalibriert werden.

Obwohl der 10-Bit ADC des Arduino einen Maximalwert von 1023 hat, ist bedingt durch den Einsatz des Instrumentenverstärkers INA114, der maximale ADC-Wert bei 857, was einer Spannung von 4,18V entspricht.

Die mit Excel gewonnenen Gleichungen der Ausgleichsgerade für Messschaltung A (siehe Abbildung 3.1) und B (siehe Abbildung 3.1) werden verwendet um aus den ADC-Werten die dazugehörigen Widerstandswerte zu berechnen.

$$R_A(ADC) = \frac{ADC_A + 4039,9}{40,107} \quad (3.1)$$

$$R_B(ADC) = \frac{ADC_B + 4023,6}{40,027} \quad (3.2)$$

3.2. Messzelle

Der Kurzschlussstrom der Messzelle wurde im Flasher (siehe Abbildung 3.3) über einen weiten Temperaturbereich gemessen (siehe Abbildung 3.4). Es wurde mit der Berger Messlast [12] gemessen. Bei der Messung wurden die Leitungswiderstände der Messkabel minimiert, indem Kabel

R / Ω	ADC Wert
101	12
102	53
103	93
104	133
105	173
106	212
107	253
108	293
109	332
110	374
111	412
112	454
113	493
114	533
115	573
116	613
117	653
118	694
119	734
120	775
121	815
122	857

Tabelle 3.1.: Die Messwerte der Kalibrierung Messschaltung A

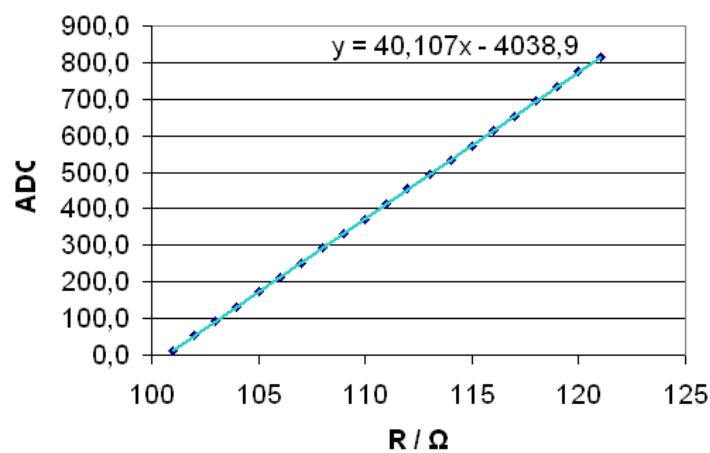


Abbildung 3.1.: Ausgleichsgerade Temperaturmessung A

R / Ω	ADC Wert
100	0
101	20
102	60
103	99
104	139
105	179
106	219
107	259
108	299
109	339
110	379
111	420
112	460
113	499
114	539
115	579
116	619
117	659
118	700
119	740
120	780
121	820
122	857

Tabelle 3.2.: Die Messwerte der Kalibrierung Messschaltung B

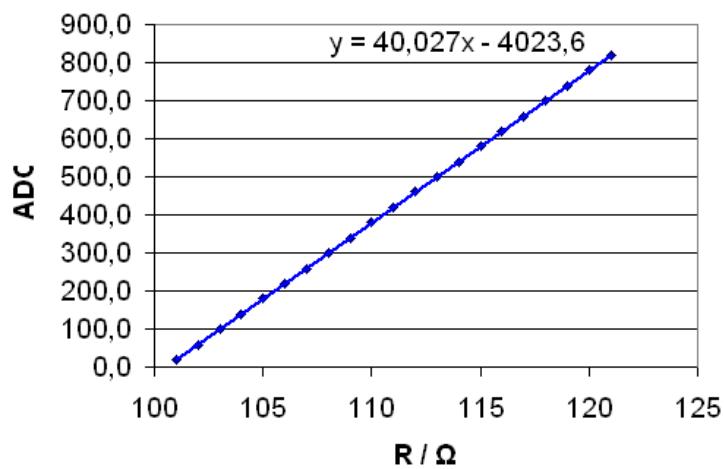


Abbildung 3.2.: Ausgleichsgerade Temperaturmessung B

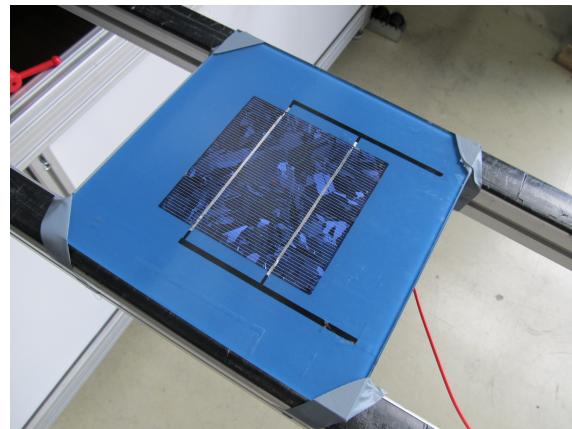


Abbildung 3.3.: Ein Photo der Messzelle

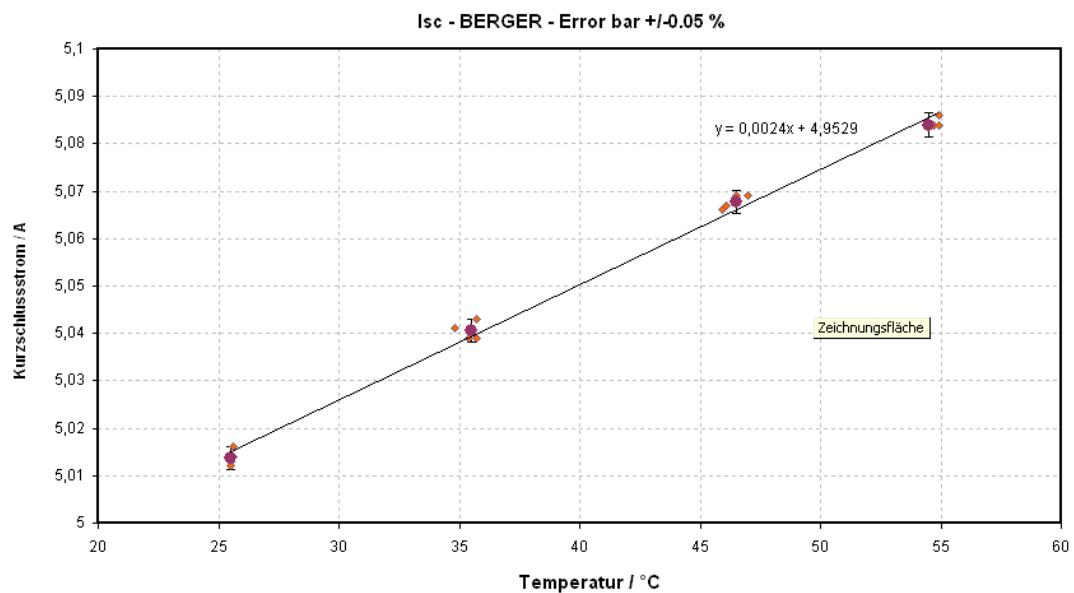


Abbildung 3.4.: Abhangigkeit des Kurzschlussstromes von der Temperatur

mit hohem Querschnitt (6mm^2) verwendet wurden, und die Kabel möglichst kurz gehalten wurden ($<1,5\text{m}$).

$$I(T) = 4,9529 + 0,0024 * T \quad (3.3)$$

$$I(T) = 4,9529 * (1 + 0,00048 * T) \quad (3.4)$$

3.3. Strommessung

Der Messroboter wurde für die Kalibrierung der Strommessung in eine Klimazelle platziert. Die Temperatur der Klimazelle wurde im Bereich von 8 bis 35°C variiert. Durch den Messwiderstand der Strommessung wurde ein Strom im Bereich von 3 bis 8 A geschickt.

T/ $^\circ\text{C}$	I/mA	ADC Wert
24,4	3997	472
	6001	712
	8002	854
35,3	3999	473
	6001	712
	7999	859
35,3	3999	472
	6001	710
	8001	852
16,4	2004	234
	3005	353
	4003	473
	5003	591
	6005	711
8	1993	233
	2995	352
	4001	472
	5000	590
	6005	710

Tabelle 3.3.: Kalibrierung Strommessung

$$I(ADC) = \frac{ADC + 4,5766}{119,2} \quad (3.5)$$

3.4. Thermische Stabilität der Temperaturmessung

Zur Bestimmung der Temperaturstabilität wurde der gesamte Roboter in ein Klimakammer gestellt. Der Pt100 wurde durch einen temperaturstabilen 110Ω simuliert. Es gab zwei Durchgänge dieses Versuches. Bei der ersten Messung zeigte sich, dass eine der beiden Messplatinen nicht temperaturstabil war. Durch das Tauschen der Ref200 Stromquelle dieser Platine konnte das Problem gelöst werden.

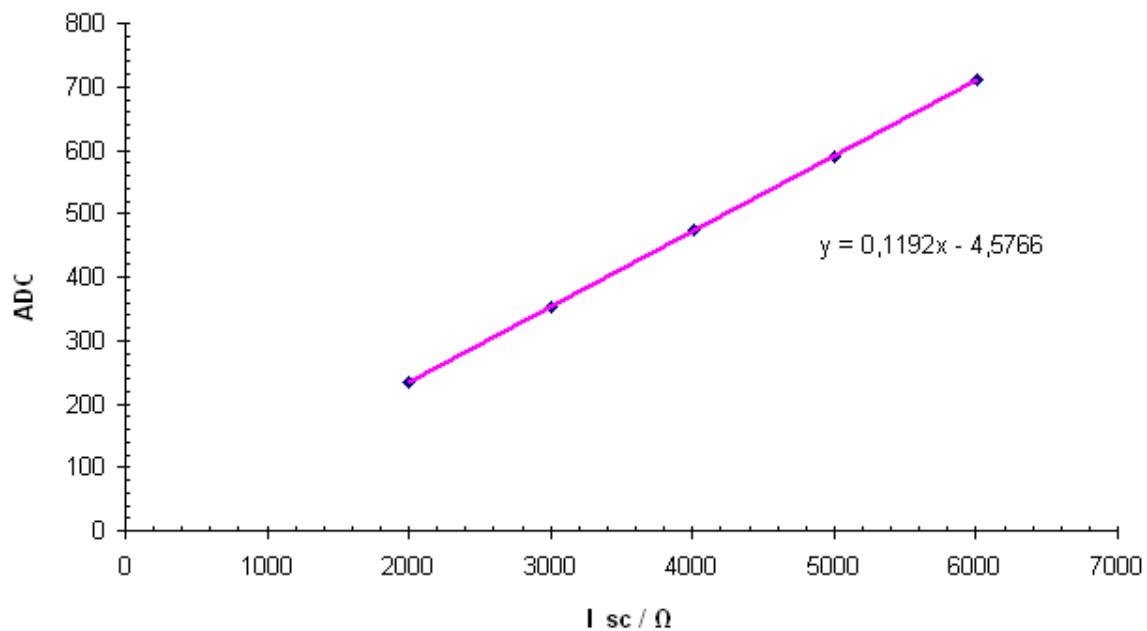


Abbildung 3.5.: Lineraer Zusammenhang zwischen Strom und dem ADC-Wert

T / °C	ADC Platine A	ADC Platine B
11,7	368	407
13,0	368	406
13,7	368	406
14,5	368	405
15,5	368	405
16,6	368	405
17,5	368	404
23,5	368	403
27,5	368	402

Tabelle 3.4.: Temperaturabhängigkeit der Temperaturmessung

T in °C	ADC Platine A	ADC Platine B
30,7	368	380
20,1	368	379
9,2	368	378

Tabelle 3.5.: Temperaturabhängigkeit der Temperaturmessung

Durch den Austausch der Referenzstromquelle der Platine B hat sich auch der ADC-Wert gegenüber der ersten Messung geändert. Die Kalibrierwerte der Platine B 3.2 sind nach diesem Austausch aufgenommen worden. Da die Platine A temperaturstabilier als die Platine B ist, wurde Platine A zur Messung der Zelltemperatur ausgewählt, die Platine B wurde für die Messung der Umgebungstemperatur verwendet.

4. Messung

Diese Kapitel beschreibt die Messvorbereitung, den Messaufbau, die eigentliche Messung und die Auswertung der Messdaten. Um was geht es? Um die Messung. Zur Messung gehört der Messaufbau, der Messablauf und die Auswertung der Messung. Weiters werden Messungen bei verschiedenen Einstellungen diskutiert.

4.1. Messaufbau

Dieser Abschnitt behandelt den Messaufbau. Die Messung der Bestrahlungsstärkeverteilung findet im Sonnensimulator statt. Zuerst muss eine Messebene geschaffen werden. Die, im normalen Messalltag vermessenen, Module liegen auf verschiebbaren Metallschienen auf. Damit der Roboter fahren kann, braucht er eine ebene Fläche. Diese Fläche ist aus logistischen Gründen aus drei einzelnen Holzplatten gebildet, was einige zusätzliche Probleme bereitet hat. Auf der Oberseite der Platten befindet sich die Messpunkte und eine Führungslinie, welcher der Roboter folgt. Zur Messung wird der Roboter an das linke Ende der Linie gestellt, er fährt nach dem Einschalten selbständig das Messprogramm ab.

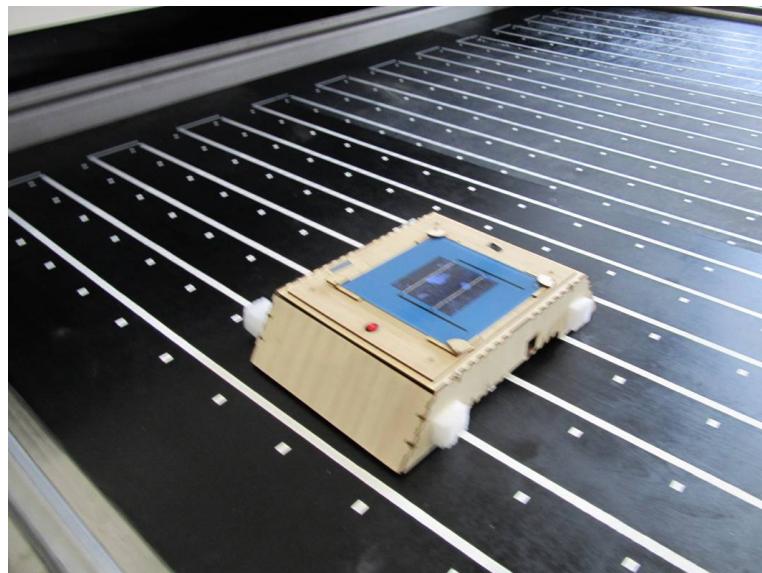


Abbildung 4.1.: Eine Fahrt des Roboters auf der Messbahn

4.1.1. Platten

Drei Holzplatten bilden die Messebene für den Roboter. Eine große Platte wäre eine bessere Alternative gewesen, aber aus logistischen Gründen ist die Messebene auf drei Platten aufgeteilt. Weil für die drei Platten leichter ein Platz zum Aufbewahren gefunden werden kann. Die 250 mal

125 Zentimeter großen und 18 Millimeter dicken Holzplattenlatte sind gerade noch hantierbar. Die Platten sind schwarz, damit sie einen möglichst geringen Reflexionskoeffizienten haben. Um einen großen Unterschied im Reflexionskoeffizienten zu haben, ist die Messbahn weiß gehalten. Sowohl der schwarze Untergrund als auch die weiße Bahn ist mit Kunstharzfarbe ausgeführt. Beim Hantieren mit den Platten können kleine Beschädigungen der Messbahn entstehen. Da diese Beschädigungen den Messablauf stören können, ist es ratsam vor der Messung die Bahn mit einem weißen Klebeband (Isolierband) auszubessern. Die Platten müssen eng aneinander anliegen, damit der Roboter von einer Platte zur anderen fahren kann. Damit keine zu groß Stufe wegen unterschiedlicher Duschbiegung der Platten entsteht ist auf der Rückseite der Holzplatten Aluminiumbleche befestigt, auf einer Seite angeschraubt, die danebenliegende Platte liegt darauf. Da die Breite aller 3 Platten geringer ist als die Breite der Lade des Sonnensimulators, werden, damit die Messungen zu verschiedenen Zeitpunkten verglichen werden können, die Platten so weit wie möglich nach rechts geschoben. Am Rand liegen die Platten auf Aluschienen auf. Die Messpunkte liegen im Raster von 15 Zentimeter. Die Führungslinie 25,4 Millimeter daneben, gemessen vom Zentrum des Messpunktes zur Mitte der Linie. Die Messpunkte sind kleine Quadrate der Seitenlänge von 14 bis 15 Millimeter. Die Messbahn ist ebenfalls 14 bis 15 Millimeter breit. Die Position der Messpunkte ist auf etwa ein bis zwei Millimeter genau. Die Messpunkte liegen im Abstand von 15 cm, damit ist auch der Abstand zwischen den Messbahnen 15 cm, der Roboter muss nur 15 cm seitwärts fahren. Die Übergänge der Messbahn von einer Platte zur nächsten sind mit weißem Isolierband der Breite 14 Millimeter zu überbrücken. Beschädigung der Messbahn, welche durch unsanftes Handling der Platten entstehen können, sind ebenfalls mit weißem Isolierband zu Überkleben. Die Messebene wird aus 3 Holzplatten gebildet, diese sind auf der Oberseite schwarz gefärbt und mit einer weißen mäanderförmigen Spur versehen. Die Platten hängen durch, dadurch ist die Messebene nicht eben. Schlimmer ist der nicht gleiche Durchhang verschiedener Platten. So kommt es zu einem Höhenunterschied zwischen den Platten. Um diese Stufen zu verringern wurde auf der Unterseite der Platten mit dem geringeren Durchhang eine kleine Aluplatte befestigt um eine Auflage für die andere Platte zu schaffen. Eine zu große Stufe behindert den Roboter beim Fahren. Weil an der Stufe nicht alle Haltepunkte zuverlässig erkannt werden können.

4.1.2. Ablauf

Vor der Messung ist der Akku des Roboters voll zu laden. Der Roboter ist auf mechanische Beschädigung zu untersuchen, eventuell ist ein Rad locker, dann lässt es sich entlang der Achse verschieben. Zuallererst sind die Platten auf Beschädigung zu untersuchen. Beschädigungen der Messbahn sind mit einem weißen Isolierband zu überkleben. Beschädigungen des Holzes direkt neben der Messbahn sind mit schwarzer Farbe auszubessern. Stellen, die der Sensor nicht sehen kann, sind egal. Die Platten sind in den Einschub des Sonnensimulators zu legen. Das ist ein Puzzle und nicht schwer. Beginnend rechts. Die Übergänge der Messbahn von einer Platte zur nächsten sind mit weißem Isolierband der Breite 14mm zu überbrücken. Beschädigung der Messbahn, welche durch unsanftes Handling der Platten entstehen können, sind ebenfalls mit weißem Isolierband zu Überkleben.

Roboter wird auf das linke vordere Ende der Messbahn gestellt. Der Roboter muss dabei auf der Messbahn stehen. Kleinere Abweichungen der Idealposition werden beim losfahren korrigiert. Vor der eigentlichen Messfahrt ist eine Testfahrt mit geöffneter Lade des Sonnensimulators zu empfehlen. Besonders die Übergänge von einer Platte zur nächsten können Probleme schaffen. Nach der bestandenen Testfahrt wird der Sonnensimulator eingeschaltet. Bis zur Messung sind 20 bis 30 Minuten zu warten, bis die Temperaturen im Simulator stabil sind. Der Roboter kann

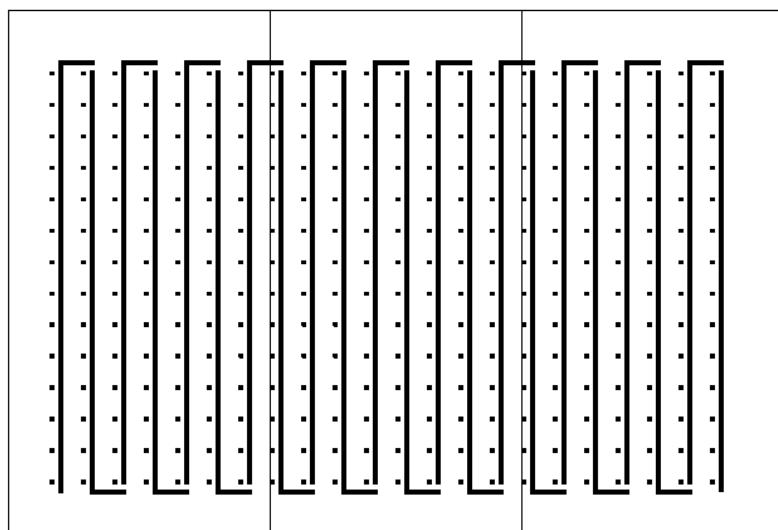


Abbildung 4.2.: Messbahn auf den Platten

während dieser Zeit im Sonnensimulator stehen. Die Messzelle sollte allerdings abgeschattet werden, damit diese sich nicht aufheizt. Zum Starten wird der Schalter von 0 auf 1 umgelegt. Damit wird der Mikrocontroller mit Spannung versorgt. Das Programm startet. Nach einer Pause von 60 Sekunden, die zum Schließen des Sonnensimulators notwendig ist, fährt der Roboter los. Eine Messfahrt dauert etwa 13 Minuten. Nach Ablauf dieser Zeit wird der Roboter entnommen und die Daten der SD-Karte entnommen, oder der Roboter wird auf die Startposition gestellt um weitere Messfahrten durchzuführen.

4.2. Auswertung

Die Messwerte werden auf SD-Karte als Textdatei gespeichert. Für jeden Messwert gibt es eine eigene Zeile. In jeder Zeile sind Messzeit in Millisekunden, der ADC-Wert für den Kurzschlussstrom, der ADC Wert für die Zelltemperatur, der ADC Wert für die Umgebungstemperatur, und jeweils einen Zählerwert für Messpunkt, Reihe und Spalte. Die einzelnen Werte sind durch einen Tabulator getrennt, damit ist die Datei in Excel und Matlab bearbeitbar.

Die Tabelle 4.1 zeigt wie die Daten auf der SD-Karte gespeichert sind. Es sind nicht alle Werte für die Auswertung notwendig. Der Erste Wert ist die Zeit in Millisekunden, die seit dem Einschalten des Roboters vergangen ist. Die nächsten 3 Werte sind werte des Analogdigitalwandlers im Bereich von Null bis 1023, diese Werte werden ohne Umrechnung direkt auf die SD Karte gespeichert. Es sind 308 Messpunkte, 14 mal 22, es kann vorkommen, am Übergang von einer Platte zur anderen, dass ein Messpunkt ausgelassen wird. Das ist erkennbar wenn nicht 308 Messwerte im datalog-File sind. Um den Fehler leicht zu finden wurde Spalte und Reihe mit aufgezeichnet. Sind in einer Spalte nur 13 Messwerte, fehlt in dieser Spalte ein Wert. Anhand der mit aufgezeichneten Zeit kann der fehlende Messpunktes lokalisiert werden. Das geschieht nicht automatisch. Der Messwert wird anhand der neben liegenden Werte geschätzt. Alternativ werden nur Messfahrten mit allen Punkten ausgewertet.

$\frac{t}{ms}$	I_{SC}	T_{Zelle}	$T_{Umgebung}$	Messpunkt	Messpunkt 2	Messreihe
165515	466	437	237	47	5	4
167523	479	437	235	48	6	4
169551	471	437	228	49	7	4

Tabelle 4.1.: Ein Ausschnitt der Messwertedatei

Im Zuge der Auswertung wird ein eindimensionales Array von 308 Messwerten in ein zweidimensionales Array von 14 mal 22 Zahlen umgewandelt. Dazu wird der erste bis zum 14. Wert des eindimensionalen Arrays zur ersten Reihe des zweidimensionalen Arrays. Der 15. bis zum 28. Wert des eindimensionalen Arrays bilden die zweite Reihe des zweidimensionalen Arrays, allerdings in umgekehrter Reihenfolge. So bildet die Reihenfolge der Messwerte des zweidimensionalen Arrays die Messfahrt des Roboters ab.

$$\begin{pmatrix} n_1 \\ \cdot \\ \cdot \\ \cdot \\ n_{308} \end{pmatrix} \Rightarrow \begin{pmatrix} n_1 & \cdot & \cdot & \cdot & n_{14} \\ n_{28} & \cdot & \cdot & \cdot & n_{15} \\ n_{29} & \cdot & \cdot & \cdot & n_{42} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ n_{295} & \cdot & \cdot & \cdot & n_{308} \end{pmatrix} \quad (4.1)$$

Um die eindimensionale Datenwurst Datenwurst in ein zweidimesionales Array, das der realen Anordnung der Messpunkte entspricht, wird folgender Algorithmus verwendet:

```
j=1; k=1; l=1;

for(i=0:307)
    if(mod(i,28)>13) k = 28- mod(i,28);
    else k = mod(i,14)+1;
    end
    B(k,l)=A(i+1);
    if(mod(i+1,14)==0) l=l+1;
    end
end
```

Beschreibung verbal: Es gibt 308 Messwerte. Die ersten 14 werden in die erste Reihe des Arrays gelegt. Die nächsten 14 in die zweite Reihe, aber in absteigender Reihenfolge. Das heißt in die 14. Spalte der 2. Reihe wird der 15. Wert gelegt, also erste Reihe von links nach rechts, die zweite Reihe von rechts nach links, die dritte Reihe wieder von links nach rechts, usw.

Der ADC Wert des Kurschlussstromes wird in Ampere umgerechnet. Verwendet wird dazu die in im Abschnitt Kalibrierung gewonnene Formel.

4.3. Vermessung der Ausleuchtung einzelner Lampen

Die geringe Messzeit des Roboters, verglichen mit der mechanischen Methode, macht es möglich die Bestrahlungsstärkeverteilung der einzelnen Lampen in realistischer Zeit zu bestimmen. Veränderbar sind der Abstand zwischen Lampenfeld und Prüfebene. Die Leistungen der Lampen lassen sich einzeln steuern. Die Lampen wurden einzeln mit einer Lampenleistung von 100% vermessen,

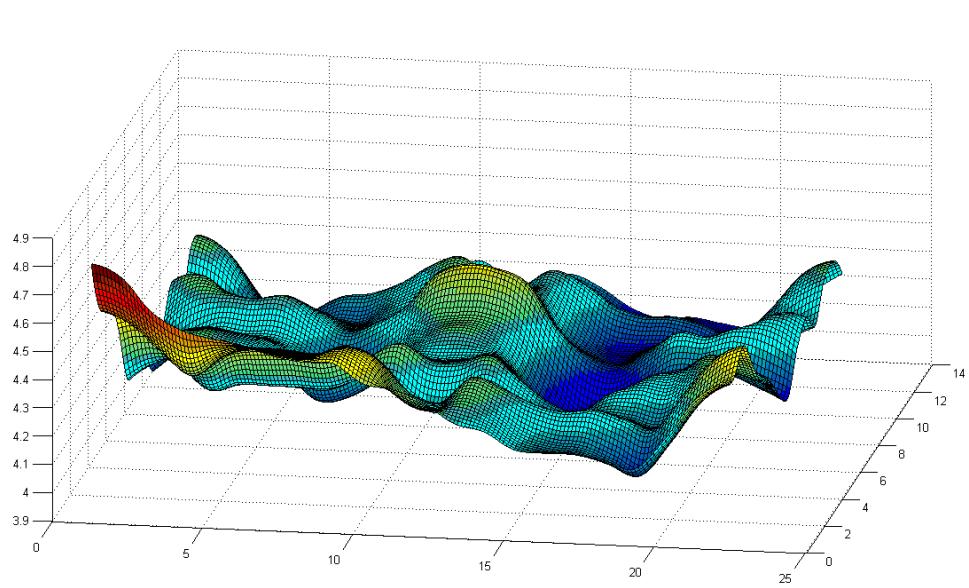


Abbildung 4.3.: Verteilung des Kurzschlusstromes in Ampere abhängig von der Position

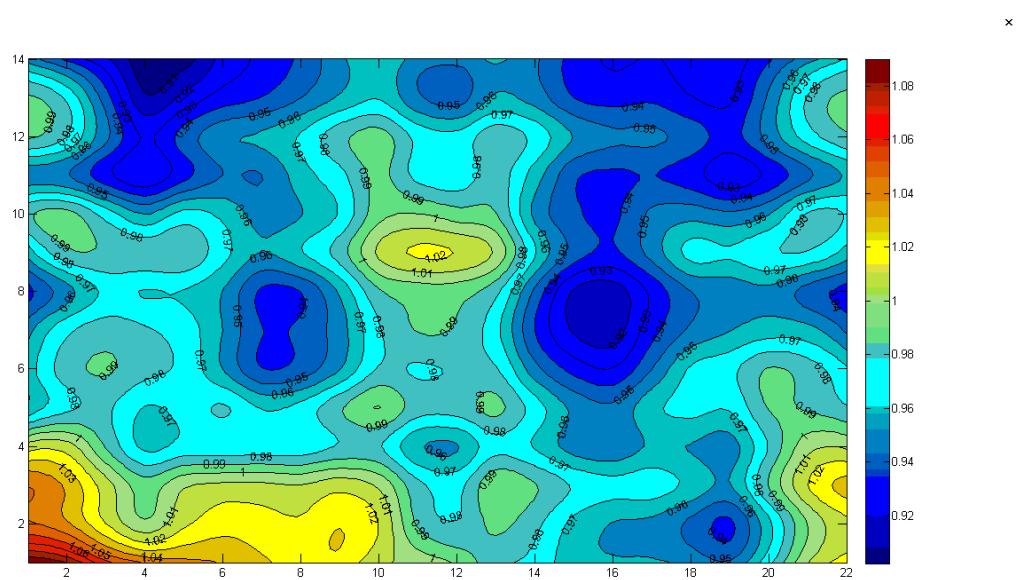


Abbildung 4.4.: Relative Verteilung der Einstrahlung

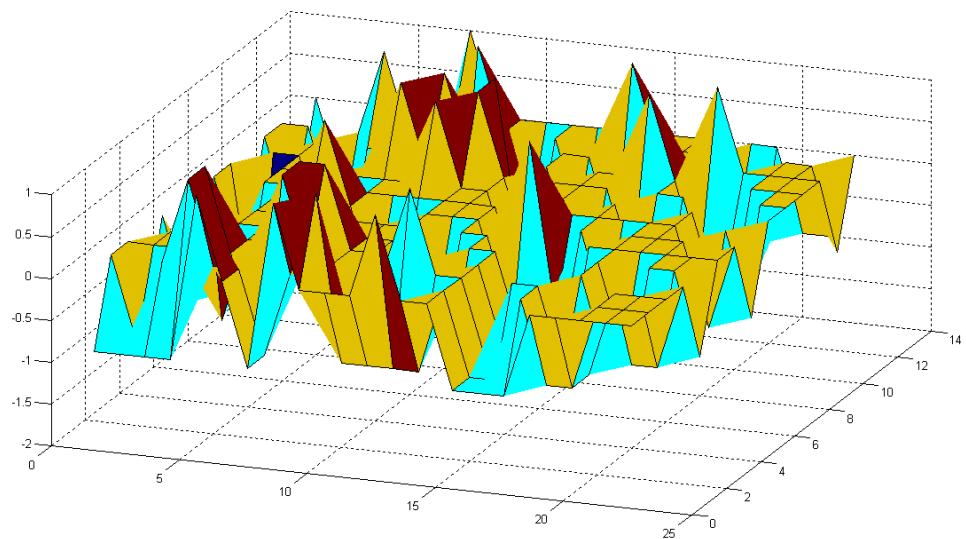


Abbildung 4.5.: Vergleich zweier Messungen

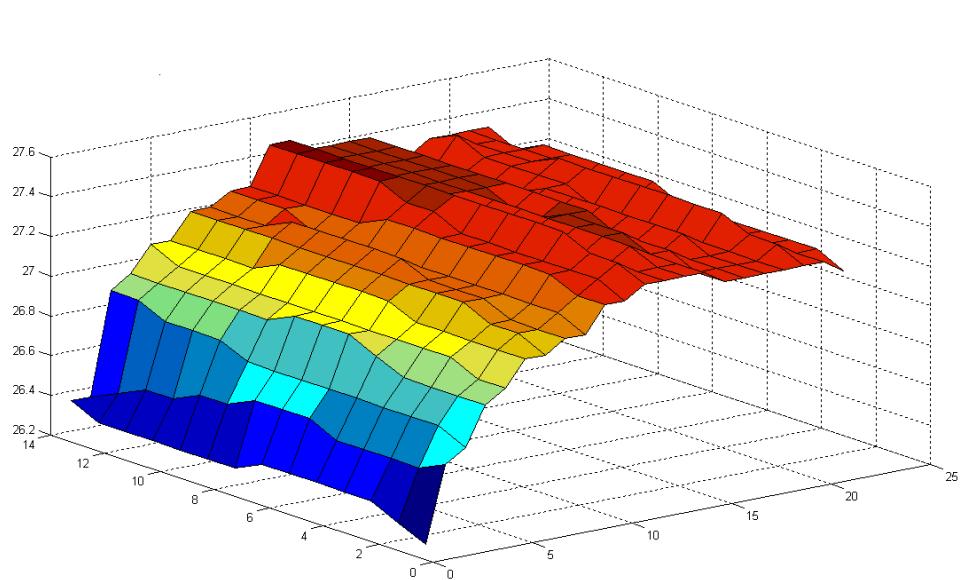


Abbildung 4.6.: Temperatur der Zelle

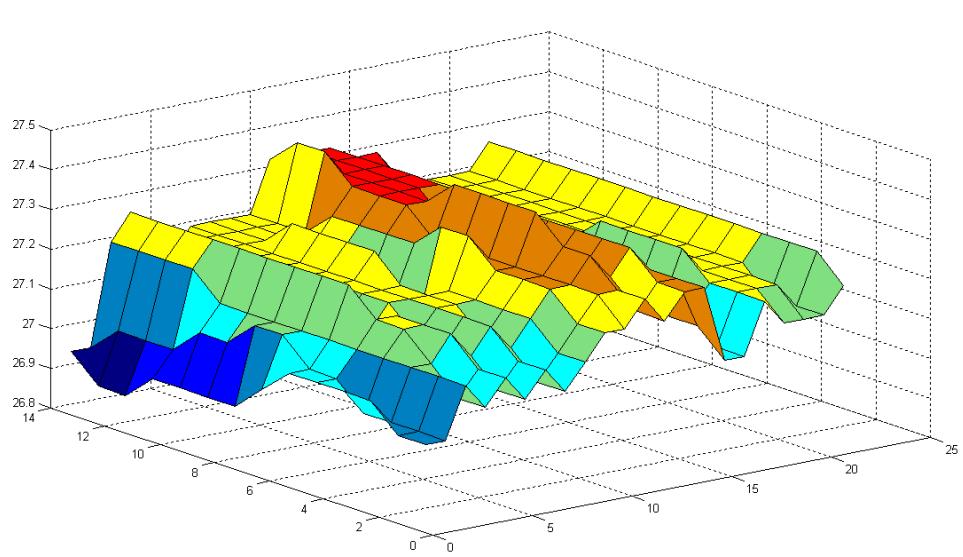


Abbildung 4.7.: konstante Temperatur der Zelle

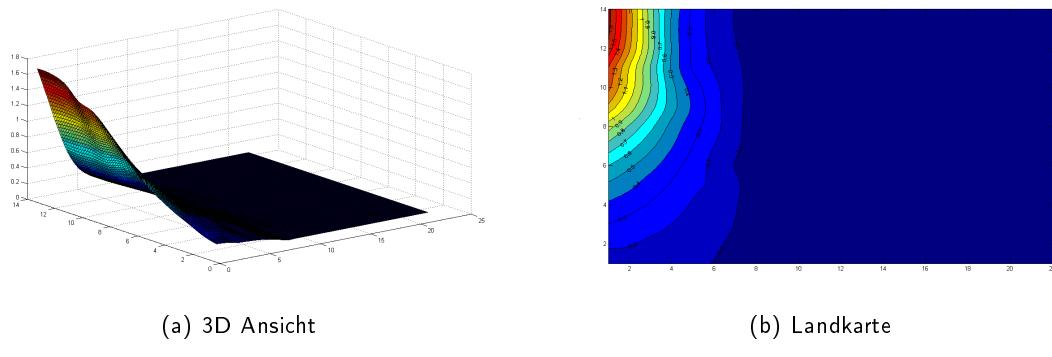
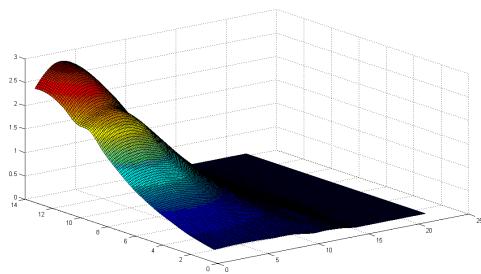


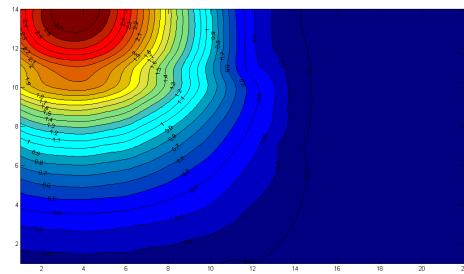
Abbildung 4.8.: Lampe E1

die anderen Lampen waren jeweils auf 0%. Aus den Werten der Einzelmessung wurde optimierte Gesamtausleuchtung errechnet. Diese optimierte Einstellung (welche noch zu messen ist) wird den Herstellerangaben verglichen. Dann wird sich zeigen ob die Methode erfolgsversprechend ist. Einfacher Algorithmus für Matlab: - Leistung einzelner Lampen nur zwischen 100 und 80- Eine der mittleren Lampen auf 100% festsetzen, andere Lampen in einem Bereich von 80% bis 100% variierten lassen. Nicht alle Lampen alle 20 Schritte machen lassen. 20 hoch 9 wären sehr viele Rechenschritte.

Zuerst muss eine Lampe bei verschiedenen Einstellungen (100, 90, 80 %) gemessen werden, um den Zusammenhang von Einstellung der Lampenleistung mit der gemessenen Bestrahlungsstärke zu ermitteln.

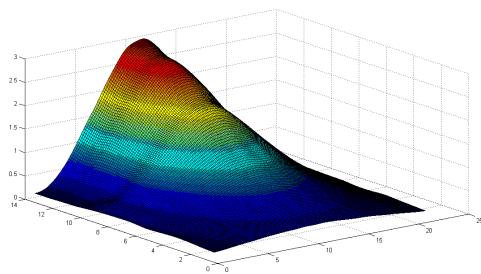


(a) 3D Ansicht

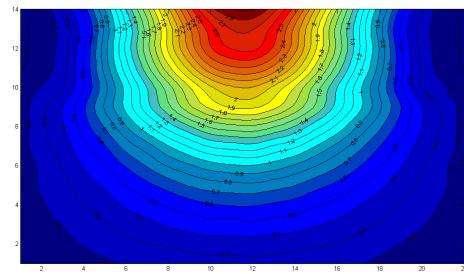


(b) Landkarte

Abbildung 4.9.: Lampe E2

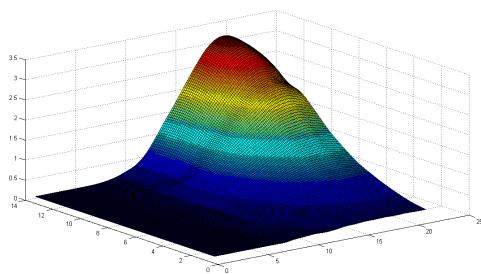


(a) 3D Ansicht

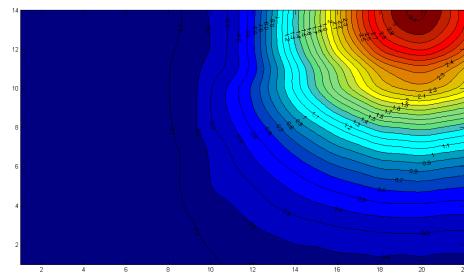


(b) Landkarte

Abbildung 4.10.: Lampe E3

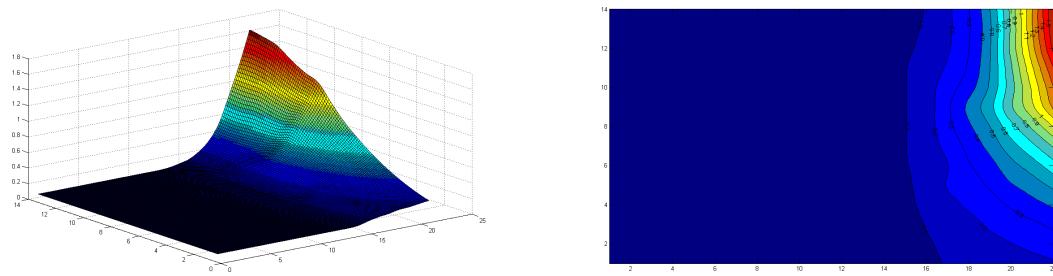


(a) 3D Ansicht



(b) Landkarte

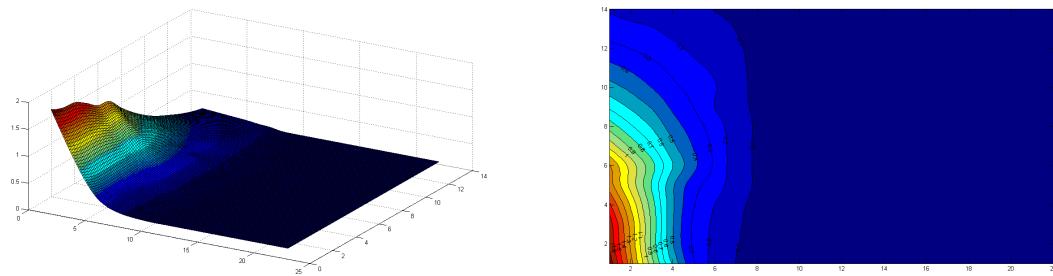
Abbildung 4.11.: Lampe E4



(a) 3D Ansicht

(b) Landkarte

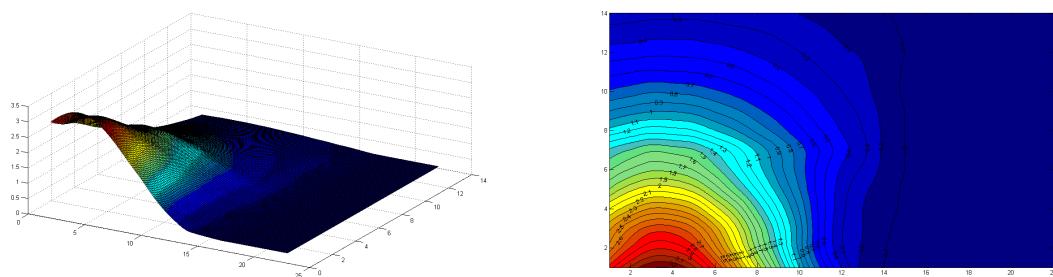
Abbildung 4.12.: Lampe E5



(a) 3D Ansicht

(b) Landkarte

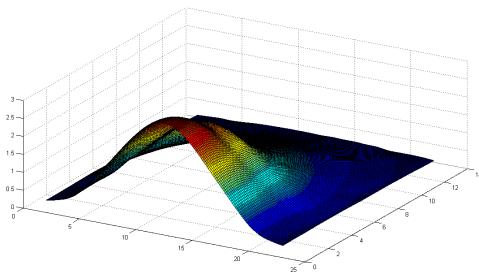
Abbildung 4.13.: Lampe E6



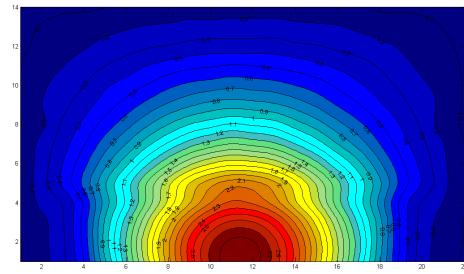
(a) 3D Ansicht

(b) Landkarte

Abbildung 4.14.: Lampe E7

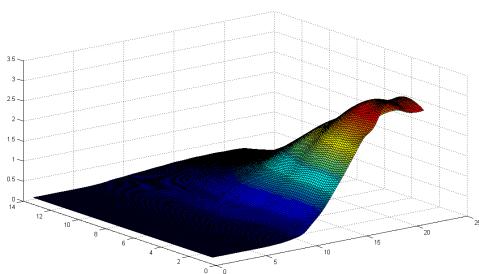


(a) 3D Ansicht

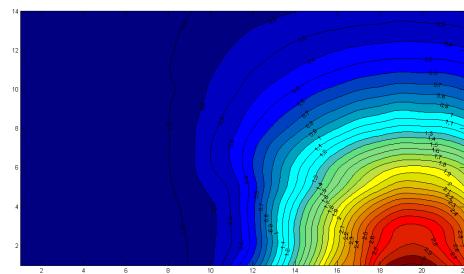


(b) Landkarte

Abbildung 4.15.: Lampe E8

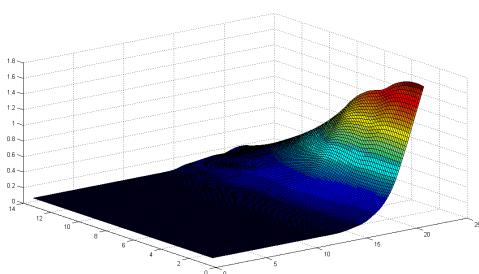


(a) 3D Ansicht

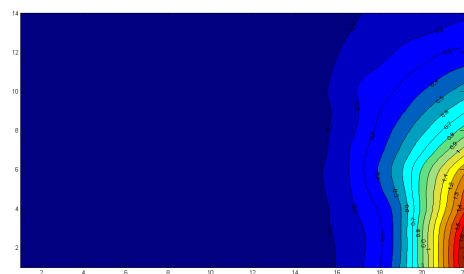


(b) Landkarte

Abbildung 4.16.: Lampe E9



(a) 3D Ansicht



(b) Landkarte

Abbildung 4.17.: Lampe E10

4.4. Schlussfolgerung

Der Roboter erleichtert die Messung der Bestrahlungsstärkemessung enorm. Durch die kurze Messdauer ist es möglich die Bestrahlungsstärke in periodischen Zeitabständen zu vermessen, um so die Alterung der Lampen zu dokumentieren. Diese Art von Messroboter kann in jedem stationären Sonnensimulator entsprechender Größe zur Vermessung der Bestrahlungsstärkeverteilung verwendet werden. Die Verwendung des Messroboters erlaubt die Bestrahlungsstärkeverteilung einzelner Lampen zu ermitteln. Ebenso die Bestrahlungsstärkeverteilung bei unterschiedlichen Höhen des Lampenfeldes.

Literaturverzeichnis

- [1] P. Würfel, *Physik der Solarzellen*, 2nd ed. Spektrum Akademischer Verlag, 2000.
- [2] "IEC 60904-3: Measurement principles for terrestrial photovoltaic (PV) solar devices with reference spectral irradiance data," IEC, 2008.
- [3] "Solar Radiation Spectrum," Bild, Wikipedia, GNU Free Documentation License, http://en.wikipedia.org/wiki/File:Solar_Spectrum.png [Zugang am 15.5.2012].
- [4] "IEC 60904-9: Solar simulator performance requirements," IEC, 2007.
- [5] "Technology Roadmaps: Solar photovoltaic energy." International Energy Agency, 2010, http://www.iea.org/publications/freepublications/publication/pv_roadmap.pdf [Zugang am 21.5.2012].
- [6] "Silicon Solar cell structure and mechanism," Bild, Wikipedia, GNU Free Documentation License, http://en.wikipedia.org/wiki/File:Silicon_Solar_cell_structure_and_mechanism.svg [Zugang am 15.5.2012].
- [7] "Solar cell equivalent circuit," Bild, Wikipedia, GNU Free Documentation License, http://en.wikipedia.org/wiki/File:Solar_cell_equivalent_circuit.svg [Zugang am 15.5.2012].
- [8] "IEC 60891: Photovoltaic devices - Procedures for temperature and irradiance corrections to measured I-V characteristics ,," IEC, 2009.
- [9] "NJM2670 - DUAL H BRIDGE DRIVER," Datenblatt, JRC.
- [10] "Arduino - PWM," <http://arduino.cc/en/Tutorial/PWM> [Zugang am 13.6.2012].
- [11] "REF200 - DUAL CURRENT SOURCE/CURRENT SINK," Datenblatt, Burr Brown, 1988.
- [12] "PSL 8 - Load and Measuring Device," Datenblatt, http://www.bergerlichttechnik.de/resources/Berger_Lichttechnik_PSL8.pdf [Zugang am 15.5.2012].

Abbildungsverzeichnis

1.1. Sonnenspektrum	2
1.2. Sonnensimulator	3
1.3. Der Schematischer Aufbau einer Siliziumsolarzelle. Quelle: [6]	5
1.4. Das Zweindiodenmodell einer Solarzelle. Quelle: [7]	5
1.5. Dunkel- und Hellkennlinien	6
1.6. Abhangigkeit des MPP von der Einstrahlung	7
2.1. Einige mogliche Fahrmanoer eines Mecanum-Rad-Roboters	9
2.2. Die Felge des Mecanum Rades	10
2.3. Ein fertig Rad, montiert am Roboter.	10
2.4. Chassis	12
2.5. Chassis	12
2.6. Die Anordnung der Elektronik auf der Bodenplatte im Roboter	13
2.7. Der Schaltplan der Motortreiberplatine.	14
2.8. Der Schaltplan des Sensorarrays.	15
2.9. Das Layout des Sensorarrays. Die optischen Sensoren sind gelb markiert.	16
2.10. Die fertige Platine des Sensorarrays.	17
2.11. Arduino Mega 2560	18
2.12. Schaltung der Temperaturmessung	19
2.13. Arduino Mega 2560	19
2.14. Die Arduino Entwicklungsumgebung	20
2.15. Die mogliche Abweichungen der Linie von der Idealposition	21
2.16. Programmablauf	22
2.17. Schranken der Regelung	22
3.1. Ausgleichsgerade Temperaturmessung A	26
3.2. Ausgleichsgerade Temperaturmessung B	27
3.3. Ein Photo der Messzelle	28
3.4. Abhangigkeit des Kurzschlussstromes von der Temperatur	28
3.5. Lineraer Zusammenhang zwischen Strom und dem ADC-Wert	30
4.1. Eine Fahrt des Roboters auf der Messbahn	32
4.2. Messbahn auf den Platten	34
4.3. Kurzschlussstromes in Ampere	36
4.4. Relative Verteilung der Einstrahlung	36
4.5. Vergleich zweier Messungen	37
4.6. Temperatur der Zelle	37
4.7. konstante Temperatur der Zelle	38
4.8. Lampe E1	38
4.9. Lampe E2	39

4.10. Lampe E3	39
4.11. Lampe E4	39
4.12. Lampe E5	40
4.13. Lampe E6	40
4.14. Lampe E7	40
4.15. Lampe E8	41
4.16. Lampe E9	41
4.17. Lampe E10	41

Tabellenverzeichnis

1.1. Spektrale Strahlungsverteilung nach IEC 60904-9	3
1.2. Anforderungen an die 3 verschiedenen Simulatorklassen	4
3.1. Die Messwerte der Kalibrierung Messschaltung A	26
3.2. Die Messwerte der Kalibrierung Messschaltung B	27
3.3. Kalibrierung Strommessung	29
3.4. Temperaturabhängigkeit der Temperaturmessung	30
3.5. Temperaturabhängigkeit der Temperaturmessung	30
4.1. Ein Ausschnitt der Messwertedatei	35

Abkürzungsverzeichnis

www World Wide Web
URL Uniform Resource Locator

A. Sourcecode Arduino

```
/*
Programm zum Steuern des Sonnensimulatormessroboters
Version 1.0
*/

#include <SD.h>

void vor(int sped);
void zuruck(int sped);
void left(int sped);
void right(int sped);
void tl(int sped);
void tr(int sped);
void ztl();
void ztr();
void vtl();
void vtr();
void halt();

// ADCs zum Auslesen der 3x3 Matrix
const int analogInPin0 = A9;
const int analogInPin1 = A2;
const int analogInPin2 = A5;
const int analogInPin3 = A8;
const int analogInPin4 = A1;
const int analogInPin5 = A4;
const int analogInPin6 = A7;
const int analogInPin7 = A0;
const int analogInPin8 = A3;
const int analogInPin9 = A6;

// Ausgang zum Schalten der Infrarot-Leds
const int analogInPin10 = A10;

// ADC für Messwerte
const int analogInPin12 = A12;
const int analogInPin13 = A13;
const int analogInPin14 = A14;
const int analogInPin15 = A15;
```

```
// Digitalausgänge zum Ansteuern der Motoren
int IN1M1 = 2;
int IN2M1 = 3;
int IN1M2 = 4;
int IN2M2 = 5;
int IN1M3 = 6;
int IN2M3 = 9;
int IN1M4 = 7;
int IN2M4 = 8;

// Sensorwerte der optischen Sensoren (beleuchtet)
int sensorValue0 = 0;           // value read from the pot
int sensorValue1 = 0;           // value read from the pot
int sensorValue2 = 0;           // value read from the pot
int sensorValue3 = 0;           // value read from the pot
int sensorValue4 = 0;           // value read from the pot
int sensorValue5 = 0;           // value read from the pot
int sensorValue6 = 0;           // value read from the pot
int sensorValue7 = 0;           // value read from the pot
int sensorValue8 = 0;           // value read from the pot
int sensorValue9 = 0;           // value read from the pot

// Sensorwerte der optischen Sensoren (unbeleuchtet)
int sensorValue0d = 0;          // value read from the pot
int sensorValue1d = 0;          // value read from the pot
int sensorValue2d = 0;          // value read from the pot
int sensorValue3d = 0;          // value read from the pot
int sensorValue4d = 0;          // value read from the pot
int sensorValue5d = 0;          // value read from the pot
int sensorValue6d = 0;          // value read from the pot
int sensorValue7d = 0;          // value read from the pot
int sensorValue8d = 0;          // value read from the pot
int sensorValue9d = 0;          // value read from the pot

// für Auswertung der opischen Sensoren
int A = 0;
int B = 0;
int C = 0;
int D = 0;
int E = 0;
int F = 0;
int G = 0;
int H = 0;
int I = 0;
int S = 0;

// Grenzwerte für hell (=white) und dunkel (=black)
```

```
int white = 400;
int white2 = 250;
int bleak = 100;
int bleak2 = 150;

// zur Berechnung der Lage der LInie
float x1,x2,x3,d,k;
float y1,y2,y3,d2,k2;

// einige Hilfsvariablen
char mode='s';
char richtung='v';
int _stop=0;
int vor_ein=0;
int zuruck_ein=0;

// für die Auswertung der Messwerte
long int temp_modul = 0;
long int temp_i=0;
long int _Isc = 0;
long int time = 0;

// zur Schreiben auf die SD Karte
const int chipSelect = 53;

// Zahlvariable für Messpunkt, Spalte und Reihe
int count = 0;
int count2 = 0;
int row = 1;

// zur Schreiben auf die SD Karte
String dataString = "";

// notwendig für die Kommunikation mit SD Karte
void setup() {
    pinMode(A10, OUTPUT);
    pinMode(53, OUTPUT);
    SD.begin(chipSelect);
}

// Definiert alle möglichen Bewegungen
void zuruck(int sped)
{
    analogWrite(IN1M1,0);
    analogWrite(IN2M1,sped);
    analogWrite(IN1M2,0);
    analogWrite(IN2M2,sped);
```

```
analogWrite(IN1M3,0);
analogWrite(IN2M3,sped);
analogWrite(IN1M4,0);
analogWrite(IN2M4,sped);
}

void vor(int sped)
{
    analogWrite(IN1M1,sped);
    analogWrite(IN2M1,0);
    analogWrite(IN1M2,sped);
    analogWrite(IN2M2,0);
    analogWrite(IN1M3,sped);
    analogWrite(IN2M3,0);
    analogWrite(IN1M4,sped);
    analogWrite(IN2M4,0);
}

void left(int sped)
{
    analogWrite(IN1M1,0);
    analogWrite(IN2M1,sped);
    analogWrite(IN1M2,sped);
    analogWrite(IN2M2,0);
    analogWrite(IN1M3,sped);
    analogWrite(IN2M3,0);
    analogWrite(IN1M4,0);
    analogWrite(IN2M4,sped);
}

void right(int sped)
{
    analogWrite(IN1M1,sped);
    analogWrite(IN2M1,0);
    analogWrite(IN1M2,0);
    analogWrite(IN2M2,sped);
    analogWrite(IN1M3,0);
    analogWrite(IN2M3,sped);
    analogWrite(IN1M4,sped);
    analogWrite(IN2M4,0);
}

void tr(int sped)
{
    analogWrite(IN1M1,sped);
    analogWrite(IN2M1,0);
    analogWrite(IN1M2,0);
```

```
analogWrite(IN2M2,sped);
analogWrite(IN1M3,sped);
analogWrite(IN2M3,0);
analogWrite(IN1M4,0);
analogWrite(IN2M4,sped);
}

void tl(int sped)
{
    analogWrite(IN1M1,0);
    analogWrite(IN2M1,sped);
    analogWrite(IN1M2,sped);
    analogWrite(IN2M2,0);
    analogWrite(IN1M3,0);
    analogWrite(IN2M3,sped);
    analogWrite(IN1M4,sped);
    analogWrite(IN2M4,0);
}

void vtr()
{
    analogWrite(IN1M1,64);
    analogWrite(IN2M1,0);
    analogWrite(IN1M2,0);
    analogWrite(IN2M2,64);
    analogWrite(IN1M3,128);
    analogWrite(IN2M3,0);
    analogWrite(IN1M4,128);
    analogWrite(IN2M4,0);
}

void vtl()
{
    analogWrite(IN1M1,0);
    analogWrite(IN2M1,64);
    analogWrite(IN1M2,64);
    analogWrite(IN2M2,0);
    analogWrite(IN1M3,128);
    analogWrite(IN2M3,0);
    analogWrite(IN1M4,128);
    analogWrite(IN2M4,0);
}

void ztr()
{
    analogWrite(IN1M1,0);
    analogWrite(IN2M1,64);
```

```
analogWrite(IN1M2,0);
analogWrite(IN2M2,64);
analogWrite(IN1M3,0);
analogWrite(IN2M3,64);
analogWrite(IN1M4,64);
analogWrite(IN2M4,0);
}

void ztl()
{
    analogWrite(IN1M1,0);
    analogWrite(IN2M1,64);
    analogWrite(IN1M2,0);
    analogWrite(IN2M2,64);
    analogWrite(IN1M3,64);
    analogWrite(IN2M3,0);
    analogWrite(IN1M4,0);
    analogWrite(IN2M4,64);
}

void halt()
{
    analogWrite(IN1M1,0);
    analogWrite(IN2M1,0);
    analogWrite(IN1M2,0);
    analogWrite(IN2M2,0);
    analogWrite(IN1M3,0);
    analogWrite(IN2M3,0);
    analogWrite(IN1M4,0);
    analogWrite(IN2M4,0);
}

// Hauptschleife
void loop()
{
    // Werte ohne Beleuchtung:
    sensorValue0d = analogRead(analogInPin0);
    sensorValue1d = analogRead(analogInPin1);
    sensorValue2d = analogRead(analogInPin2);
    sensorValue3d = analogRead(analogInPin3);
    sensorValue4d = analogRead(analogInPin4);
    sensorValue5d = analogRead(analogInPin5);
    sensorValue6d = analogRead(analogInPin6);
    sensorValue7d = analogRead(analogInPin7);
    sensorValue8d = analogRead(analogInPin8);
```

```
sensorValue9d = analogRead(analogInPin9);
digitalWrite(A10, HIGH); // LED ein
delay(2);
// Werte mit Beleuchtung:
sensorValue0 = analogRead(analogInPin0);
sensorValue1 = analogRead(analogInPin1);
sensorValue2 = analogRead(analogInPin2);
sensorValue3 = analogRead(analogInPin3);
sensorValue4 = analogRead(analogInPin4);
sensorValue5 = analogRead(analogInPin5);
sensorValue6 = analogRead(analogInPin6);
sensorValue7 = analogRead(analogInPin7);
sensorValue8 = analogRead(analogInPin8);
sensorValue9 = analogRead(analogInPin9);
digitalWrite(A10, LOW);

A = sensorValue1 - sensorValue1d;
B = sensorValue2 - sensorValue2d;
C = sensorValue3 - sensorValue3d;
D = sensorValue4 - sensorValue4d;
E = sensorValue5 - sensorValue5d;
F = sensorValue6 - sensorValue6d;
G = sensorValue7 - sensorValue7d;
H = sensorValue8 - sensorValue8d;
I = sensorValue9 - sensorValue9d;

S = sensorValue0 - sensorValue0d;

// zum Erkennen der Linie bei vor- oder zurückfahren
x1 = ((C-A)/float(2*A-4*B+2*C));
x2 = ((F-D)/float(2*D-4*E+2*F));
x3 = ((I-G)/float(2*G-4*H+2*I));

d = (x1 + x2 + x3)/3.; // ~ Abstand von der Ideallinie
k = (x1-x3)/2.; // Steigung = Verdrehung

// zum Erkennen der Linie bei links- oder rechtsfahren
y1 = ((G-A)/float(2*A-4*D+2*G));
y2 = ((H-B)/float(2*B-4*E+2*H));
y3 = ((I-C)/float(2*C-4*F+2*I));

d2 = (y1+y2+y3)/3.;
k2 = (y3-y1)/2.;

if( S < bleak2) _stop=0;

halt();
```

```
if(E>bleak2)
{

    switch(mode)
    {
        case 's': // Start
        {
            halt();
            delay(60000); // 1 Minuten warten
            mode='v';
        }
        break;

        // Messmode
        case 'e': // Ende
        {
            halt();
        }
        break;

        case 'm': // Messen
        {
            count = count + 1; // Anzahl der Haltepunkte
            count2 = count2 + 1;
            dataString = "";
            _stop=1;
            halt();

            delay(250); // damit die Strome der Fahrmotoren keinen Einfluss auf die AD-Wand

            _Isc = 0;
            temp_modul = 0;
            temp_i = 0;

            // Mittelung über jeweils 500 Messwerte
            for(int i = 0; i < 500 ; i++)
            {
                _Isc = _Isc + analogRead(analogInPin13);
                temp_modul = temp_modul + analogRead(analogInPin12);
                temp_i = temp_i + analogRead(analogInPin15);
            }
            _Isc = int(_Isc/500.0);
            temp_modul = int(temp_modul/500.0);
            temp_i = int(temp_i/500.0);
            time = millis();
        }
    }
}
```

```
// Schreiben auf SD-Karte
dataString += String(time);
dataString += "\t";
dataString += String(_Isc);
dataString += "\t";
dataString += String(temp_modul);
dataString += "\t";
dataString += String(temp_i);
dataString += "\t";
dataString += String(count);
dataString += "\t";
dataString += String(count2);
dataString += "\t";
dataString += String(row);

// Schreibe auf SD Card
File dataFile = SD.open("datalog.txt", FILE_WRITE);
if (dataFile) {
    dataFile.println(dataString);
    dataFile.close();
}

// zurück in den Bewegungsmodus
if(richtung=='v') mode='v';
if(richtung=='z') mode='z';

}
break;

case 'v': // Vorwärtsfahren
{
    richtung='v';

    if(d>0.02) vtr();
    else
    {
        if(d>-0.02) vor(128);
        else
            vtl();
    }

//if((C<bleak)&&(A<bleak))
if(A<bleak2)
{
    if(k>0.01) tr(64);
    else
```

```
{  
    if(k<-0.01) tl(64);  
}  
  
if(d>0.025) right(64);  
else  
{  
    if(d<-0.025) left(64);  
}  
}  
if((A>white)&&(B>white)) mode='a';  
  
if(_stop==0)&&(S>white2)) mode='m';  
  
}  
break;  
  
case 'z': // Rückwärtsfahren  
{  
  
richtung='z';  
  
if(d>0.02) ztr();  
else  
{  
    if(d>-0.02) zuruck(128);  
    else ztl();  
}  
  
//if((G<bleak)&&(I<bleak))  
if(G<bleak2)  
{  
    if(k>0.01) tr(64);  
    else  
{  
        if(k<-0.01) tl(64);  
    }  
}  
if(d>0.025) right(64);  
else  
{  
    if(d<-0.025) left(64);  
}  
}  
if((G>white)&&(H>white)) mode='c';
```

```
if(( _stop==0)&&(S>white2)) mode='m';

if((I<bleak)&&(H<bleak)&&(G<bleak)) mode='e';

}

break;

case 'r':      //nach rechts
{
    richtung='r';
    count2 = 0;

    right(128);
    if(D>white2)
    {
        if(d2>0.04) vor_ein=1;
        if(d2<0.02) vor_ein=0;
        if(d2<-0.04) zuruck_ein=1;
        if(d2>-0.02) zuruck_ein=0;

        if(vor_ein==1) vor(64);
        if(zuruck_ein==1) zuruck(64);

        if((vor_ein==0)&&(zuruck_ein==0))
        {
            if(k2>0.02) tr(64);
            else
            {
                if(k2<-0.02) tl(64);
            }
        }
    }

    if((D<bleak)&&(G<bleak)&&(A<bleak))
    {
        mode='b';
    }

}
break;

case 'l':      // auch nach rechts
{
    richtung=='l';
    count2 = 0;

    right(128);
```

```
//if((C<bleak)&&(I<bleak))
if(D>white2)
{
    if(d2>0.04) vor_ein=1;
    if(d2<0.02) vor_ein=0;
    if(d2<-0.04) zuruck_ein=1;
    if(d2>-0.02) zuruck_ein=0;

    if(vor_ein==1) vor(64);
    if(zuruck_ein==1) zuruck(64);

    if((vor_ein==0)&&(zuruck_ein==0))
    {
        if(k2>0.02) tr(64);
        else
        {
            if(k2<-0.02) tl(64);
        }
    }
}

if((D<bleak)&&(G<bleak)&&(A<bleak))
{
    mode='d';
}

}
break;

case 'a': // Um die Ecke fahren
{
    if((A>bleak2)&&(B>bleak2)) vor(64);
    else
    {
        right(64);
        delay(700);
        halt();
        row = row +1 ;

        mode='r';
    }
}
break;

case 'b': // Um die Ecke fahren
{
```

```
zuruck(64);
delay(850);
halt();
delay(500);

mode='x';

}

break;

case 'x':
{
    if(d>0.025) right(64);

    if(d<-0.025) left(64);

    if((d<0.1)&&(d>-0.1)) mode='y';
}
break;

case 'y':
{
    if(k>0.01) tr(32);

    if(k<-0.01) tl(32);

    if((k<0.04)&&(k>-0.04)) mode='z';
}
break;

case 'c': // Um die Ecke fahren
{
    if((H>bleak2)&&(G>bleak2 )) zuruck(64);
    else
    {
        right(64);
        delay(700);
        halt();
        row = row +1 ;

        mode='l';
    }
}
break;

case 'd': // Um die Ecke fahren
```

```
{  
  
    vor(64);  
    delay(850);  
    halt();  
    delay(500);  
    mode='q';  
  
}  
break;  
  
case 'q':  
{  
    if(d>0.025) right(64);  
  
    if(d<-0.025) left(64);  
  
    if((d<0.1)&&(d>-0.1)) mode='p';  
}  
break;  
  
case 'p':  
{  
    if(k>0.01) tr(32);  
  
    if(k<-0.01) tl(32);  
  
    if((k<0.04)&&(k>-0.04)) mode='v';  
}  
break;  
}  
}  
  
else  
{  
    if((richtung=='v')||(richtung=='z'))  
    {  
        if((A>bleak2)|| (D>bleak2)|| (G>bleak2)|| (C>bleak2)|| (F>bleak2)|| (I>bleak2))  
        {  
            if((A+D+G)>(C+F+I)) right(128);  
            else left(128);  
        }  
        else halt();  
    }  
}
```

```
if((richtung=='r')||(richtung=='l'))
{
    if((A>bleak2)|| (B>bleak2)|| (C>bleak2)|| (G>bleak2)|| (H>bleak2)|| (I>bleak2))
    {
        if((A+B+C)>(G+H+I)) vor(64);
        else zuruck(64);
    }
    else halt();
}

delay(5);

}
```

B. Sourcecode Auswertung

```
% Graphische Auswertung der Robotermesswerte

close all;
clc;
clear all;

% "Offnen der Datei, welche eine Messfahrt mit genau 308 Messwerten
% enthalten muss
fid = fopen('data1.txt', 'r');
a = fscanf(fid, '%g %g', [7 308])
a = a';
fclose(fid)

j=1; k=1; l=1;

% Umwandeln der 1 dimensionalen Modultemperatur ADC-Werte in eine Matrix:
for(i=0:307)
    if(mod(i,28)>13) k = 28- mod(i,28);
    else k = mod(i,14)+1;
    end
    Tm(k,l)=a(i+1,3);
    if(mod(i+1,14)==0) l=l+1;
    end
end

% Umrechnung der ADC-Werte in Temperatur:
Rm = (Tm +4038.9)/40.107; % ADC --> Widerstand
TTm = (Rm-100.03)/0.3879; % Widerstand --> Temperatur

j=1; k=1; l=1;

% Umwandeln der 1 dimensionalen Innentemperatur ADC-Werte in eine Matrix:
for(i=0:307)
    if(mod(i,28)>13) k = 28- mod(i,28);
    else k = mod(i,14)+1;
    end
    Ti(k,l)=a(i+1,4);
    if(mod(i+1,14)==0) l=l+1;
    end
```

```

end

% Umrechnung der ADC-Werte in Temperatur:
Ri = (Ti +4023.6)/40.027; % ADC --> Widerstand
TTi = (Ri-100.03)/0.3879; % Widerstand --> Temperatur

j=1; k=1; l=1;

% Umwandeln der 1 dimensionalen Kurzschlusstrom ADC-Werte in eine Matrix:
for(i=0:307)
    if(mod(i,28)>13) k = 28- mod(i,28);
    else k = mod(i,14)+1;
    end
    I(k,l)=a(i+1,2);
    if(mod(i+1,14)==0) l=l+1;
    end
end

% Umrechnung der ADC-Werte in Ampere:

A = (I +4.5766)/119.2; % ADC --> Ampere
A = A - (TTm-25)*0.004; % Temperaturkorrektur

[XI,YI] = meshgrid(1:.125:22, 1:.125:14);

Ai = interp2(A,XI,YI,'cubic'); % Interpolation

A_max = max(max(Ai));
A_min = min(min(Ai));

Normiert = Ai / A_max * 1.1;

w = (A_max - A_min)/(A_max + A_min) * 100 % maximale Abweichung in Prozent

% Graphische Darstellung des Stromes
figure;
surf(XI,YI,Ai); % in Ampere

% Graphische Darstellung des normierten Stromes
zlevs2 = 0.9:0.01:1.1;
figure;
[C,h] = contourf(XI,YI,Normiert,zlevs2);
set(h,'ShowText','on','TextStep',get(h,'LevelStep')*2)
colorbar;

```

```
% Graphische Darstellung der Modultemperatur
figure;
surf(TTm);

% Graphische Darstellung der Umgebungstemperatur
figure;
surf(TTi);
```