

# **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, 3. Mai 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  $\text{Pr} \frac{1}{4}$ -Ebene eines stationären Sonnensimulators erfasst. Neben den Einstrahlungsdaten werden zusätzliche Informationen über die Umgebungsbedingungen im  $\text{Pr} \frac{1}{4}$ -Kanal 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:** Schlagwort 1, Schlagwort 2, Schlagwort 3, Schlagwort 4, Schlagwort 5

## **Abstract**

Text  
Text Text Text Text Text ...

**Keywords:** Keyword 1, Keyword 2, Keyword 3, Keyword 4, Keyword 5

## **Danksagung**

Ich danke meinen Eltern fÃ¼r die UnterstÃützung und Geduld, die sie wÃ¤hrend des Studiums aufgebracht haben. Ich danke meinen Hochschulbetreuer fÃ¼r die umfangreiche Betreuung. Ich danke meinen Firmenbetreuer Thomas Krametz fÃ¼r die Zeit die er sich genommen hat. ... (Rohfassung!!!!)

# Inhaltsverzeichnis

<b>1. Aufgabenstellung</b>	<b>1</b>
1.1. Warum Module im Sonnensimulator . . . . .	1
1.2. Sonnensimulator Aufbau . . . . .	1
1.3. Notwendigkeit der Ausleuchtungsmessung . . . . .	1
1.4. Theorie Referenzzelle . . . . .	1
<b>2. Entwicklungprozess</b>	<b>2</b>
2.1. Hardwaredesign . . . . .	2
2.1.1. Mecanum-Platform . . . . .	2
2.1.2. Chassis . . . . .	2
2.2. Steuerungselektronik . . . . .	4
2.2.1. Motorensteuerung . . . . .	4
2.2.2. Optische Sensorik . . . . .	4
2.2.3. Spannungversorgung . . . . .	4
2.2.4. Mikrokontroller . . . . .	4
2.2.5. Temperatursensoren . . . . .	7
2.3. Softwareentwicklung . . . . .	7
2.3.1. Entwicklungsumgebung . . . . .	7
2.3.2. Auswertung optische sensoren . . . . .	7
2.3.3. Auswertung ADCs . . . . .	7
2.3.4. Programmablauf . . . . .	7
<b>3. Kalibration</b>	<b>8</b>
3.1. Temperatursensoren . . . . .	8
3.2. Messzelle . . . . .	8
3.3. Strommessung . . . . .	9
3.4. Thermische Stabilität . . . . .	9
<b>4. Messung</b>	<b>10</b>
4.1. Messaufbau . . . . .	10
4.1.1. Platten . . . . .	10
4.1.2. Ablauf . . . . .	10
4.2. Auswertung . . . . .	10
4.3. Schlussfolgerung . . . . .	10
<b>Abbildungsverzeichnis</b>	<b>11</b>
<b>Tabellenverzeichnis</b>	<b>12</b>
<b>Abkürzungsverzeichnis</b>	<b>13</b>

<b>A. Sourcecode Arduino</b>	<b>14</b>
<b>B. Sourcecode Auswertung</b>	<b>29</b>

# **1. Aufgabenstellung**

## **1.1. Warum Module im Sonnensimulator**

Ein Sonnensimulator simuliert die Wirkung von Sonnenlicht auf Prüfobjekte.

## **1.2. Sonnensimulator Aufbau**

10 Lampen zu je 4 kW. Angeordnet in 2 Reihen zu je 5 Lampen. Die Lampen sind höhenverstellbar. Die Leistung der Lampen lässt sich einzeln ansteuern. Die Testobjekte, z.B. ganze Module oder auch einzelne Zellen liegen in einer Lade. Die Prüfebene ist 2,50 mal 4 Meter groß. Die Prüfebene liegt innerhalb eines Windkanals, der oben für die Strahlung durchsichtig ist. Der Windkanal verengt sich, damit die Windgeschwindigkeit ansteigt, um eine konstante Kühlleistung wegen der sich erwärmenden Luft zu haben.

## **1.3. Notwendigkeit der Ausleuchtungsmessung**

Akkreditierung, Begründung warum Messroboter, Alte Ergebnisse, Alte Messmethode

## **1.4. Theorie Referenzzelle**

Referenzzelle, Temperaturabhängigkeit Messzelle, Kennlinie, Warum Kurzschlussstrom,...

## 2. Entwicklungprozess

### 2.1. Hardwaredesign

#### 2.1.1. Mecanum-Platform

Das Mecanum-Rad ist ein Rad, das Fahrmanöver in jede Richtung erlaubt, ohne dass das Fahrzeug mit einer mechanischen Lenkung ausgestattet ist. Benannt ist es nach dem schwedischen Unternehmen Macanum AB in welchen dieses Rad 1971 entwickelt wurde. Erreicht wird die Wendigkeit der Fahrzeuge durch den Einsatz von Mecanum Rädern, die einzeln angetrieben werden. Diese Räder bestehen aus einer Felge, auf der unter einem Winkel von 45 Grad lose, ballige Rollen so angebracht sind, dass sie über den Abrollumfang wieder einen exakten Kreis bilden. Durch die Schräganordnung der Rollen entstehen beim Antreiben des Rades 2 Kraftkomponenten. 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 kann durch entsprechendes Ansteuern der einzelnen Räder bezüglich Drehrichtung und -geschwindigkeit jedes beliebige Fahrmanöver erzeugt werden.

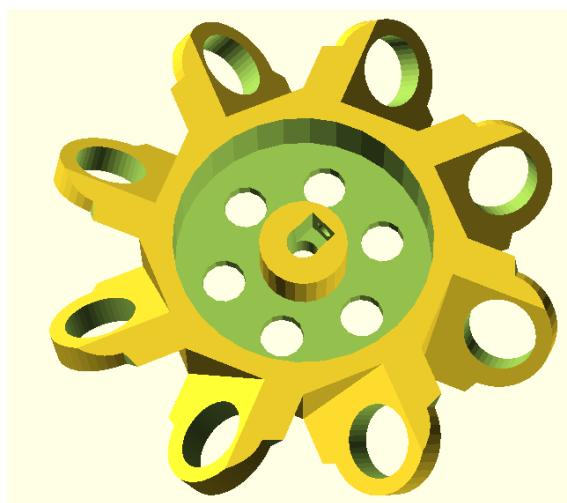


Abbildung 2.1.: Mecanum-Rad

#### 2.1.2. Chassis

Die Chassis wurde aus Verbund-Holz gefertigt. Das Design dafür entstand mit QCAD. Die Einzelteile wurden mit einem Lasercutter aus einer großen Platte geschnitten. Dann verschraubt und teilweise geleimt. Der Oberteil des Chassis lässt sich von der Bodenplatte trennen. Die Motoren sind verschraubt. Einzelne Räder und auch Rollen lassen sich bei Bedarf austauschen. Das Chassis besteht aus 3 Teilen:

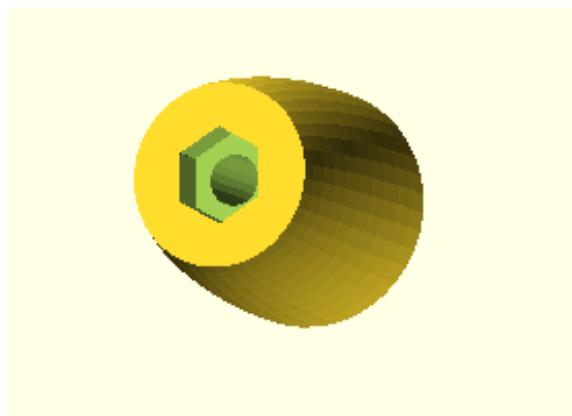


Abbildung 2.2.: Rolle

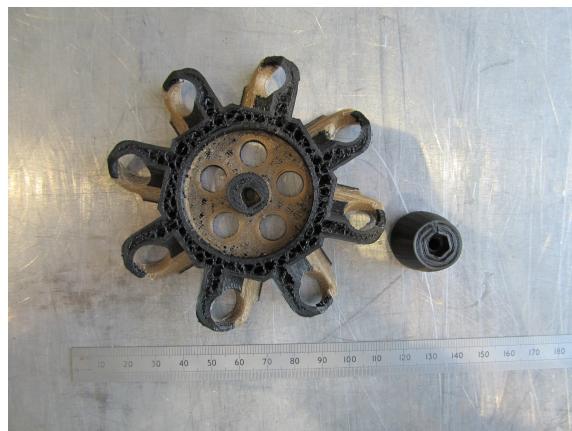


Abbildung 2.3.: Foto eines gedruckten Rades

- die Bodenplatte
- die Seitenwände
- das Dach mit der Aufnahme für die Messzelle

## 2.2. Steuerungselektronik

### 2.2.1. Motorensteuerung

H-Brücken Treiber: NJM2670 The NJM2670 is a general-purpose 60V dual H-bridge drive IC. It consists of a pair of H-bridges, a thermal shut down circuit and its alarm output. The alarm output can detect application problems and the system reliability will be significantly improved if monitored by Micro Processor. Therefore, it is suitable for two-phase stepper motor application driven by microprocessor. Motor: Getriebemotor RB 35, 1:200 Versorgungsspannung: 12V wurde auf 15,5 bis 16 erhöht Drehzahl: 30 Umdrehungen pro Minute im Leerlauf bei 12V Versorgung Achsdurchmesser: 6mm

### 2.2.2. Optische Sensorik

Der Roboter ist ein Linienfolge. Damit er einer Linie folgen kann, muss diese zuerst zuverlässig erkannt werden. Als Sensor wurde ein CNY70 verwendet. Der CNY 70 ist ein Reflexionsensor. 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 kannen Unterschiede des Reflektionkoeffizienten, umgangssprachlich der Farbe, gemessen werden. Die Anordnung 9 solcher Sensoren in einem 3 mal 3 Array kann sowohl vertikale als auch horizontale Linien erkennen, sowie Ecken.

### 2.2.3. Spannungversorgung

Aku-handling, Überwachung der Akkuspannung, Alarm und Abschaltung bei Unterspannung. Es wird empfohlen vor jeden Messzyklus den Akku zu laden. Ein voller Akku schafft ohne Probleme 10 Messfahrten. Bauteile: Akku, Ladegerät (extern), bequemes und schnelles Laden des Akkus Eine Schmelzsicherung ist direkt im Anschlusskabel des Akkus eingebaut. Der Akku schafft Strom bis 200A, da er aus dem Modellbausektor kommt. Was für den Roboter völlig überdimensioniert ist. Der maximale Strom aller 4 Motoren und der restlichen Elektronik liegt bei einem Ampere. Die Motoren werden direkt mit der Akkuspannung versorgt. Da realistischerweise die Versorgungsspannung zwischen 16,4V (Akku voll) und 15,5V (nach vielen Fahrten) liegt, ist der Einfluss auf die Drehzahl der Motoren zu vernachlässigen. Die 5 Volt Versorgung des Mikrocontrollers und der OPVs wurde mit einem RECOM R-785.0-1.0 DC-DC Konverter gelöst. Er ist pinkompatibel zu einem 7805, und bietet die Vorteile einer geringen Verlustleistung und einer sehr genauen Ausgangsspannung. Die Analog-Digitalwandlung des Arduino benötigt eine Referenzspannung. Das kann die Versorgungsspannung sein, was aber doch zu ungenau ist. The LT1021 is a precision reference with ultralow drift and noise, extremely good long term stability and almost total immunity to input voltage variations. The reference output will both source and sink up to 10mA. Three voltages are available: 5V, 7V and 10V. The 7V and 10V units can be used as shunt regulators (two-terminal zeners) with the same precision characteristics as the three-terminal connection. Special care has been taken to minimize thermal regulation effects and temperature induced hysteresis.

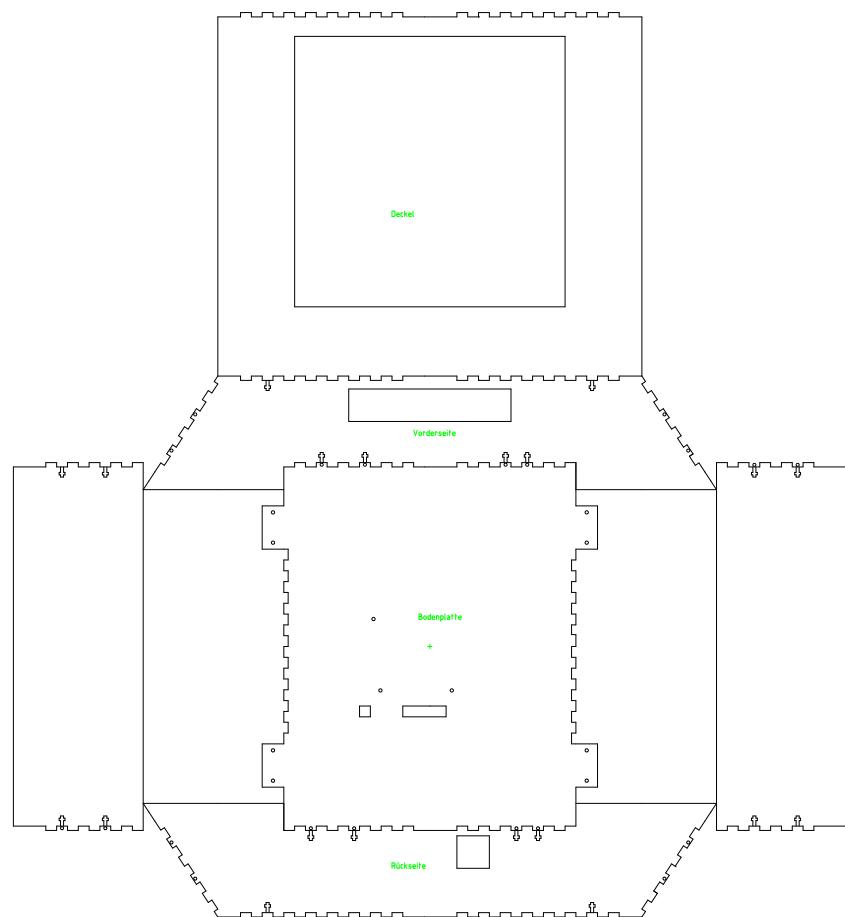


Abbildung 2.4.: Chassis

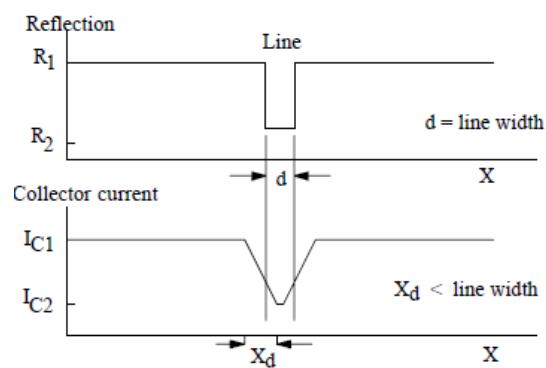


Abbildung 2.5.: ...

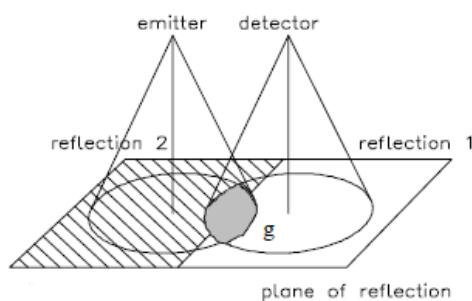


Abbildung 2.6.: ...

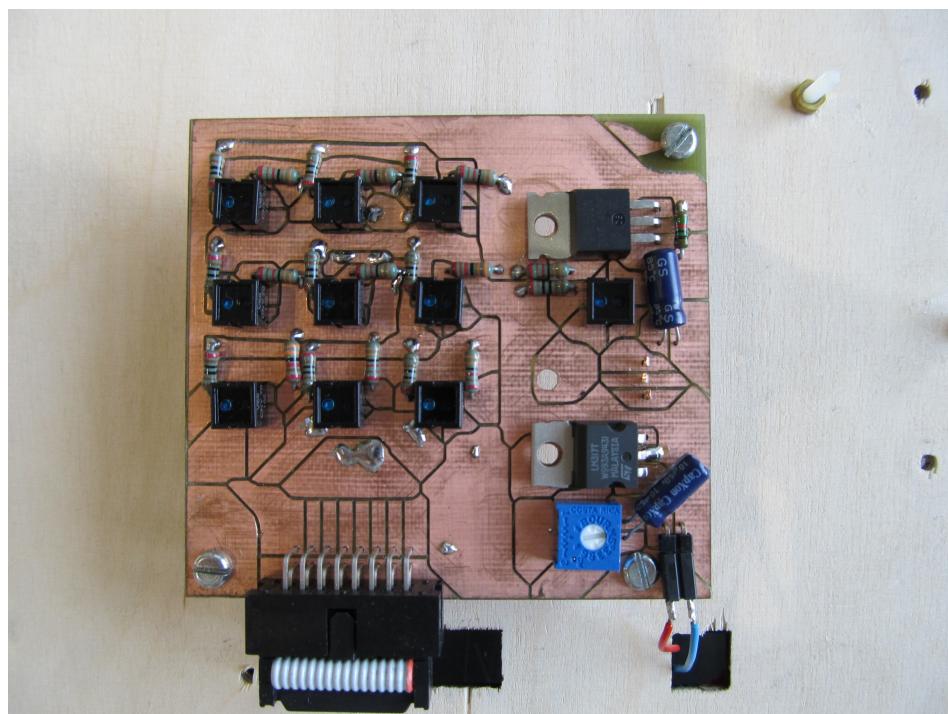


Abbildung 2.7.: Sensorarrangement im Roboter eingebaut

### **2.2.4. Mikrokontroller**

Arduino (was ist das), Shield, SD, Aref,

### **2.2.5. Temperatursensoren**

Messprinzip, Temperaturstabilität

## **2.3. Softwareentwicklung**

### **2.3.1. Entwicklungsumgebung**

### **2.3.2. Auswertung optische sensoren**

Differentielle Messung,

Mathematik Auswertung,

Folge der Linie

Behandlung der Ecken

Haltepunkte

### **2.3.3. Auswertung ADCs**

Mittelung, Rauschgrößen,

### **2.3.4. Programmablauf**

# 3. Kalibration

## 3.1. Temperatursensoren

## 3.2. Messzelle

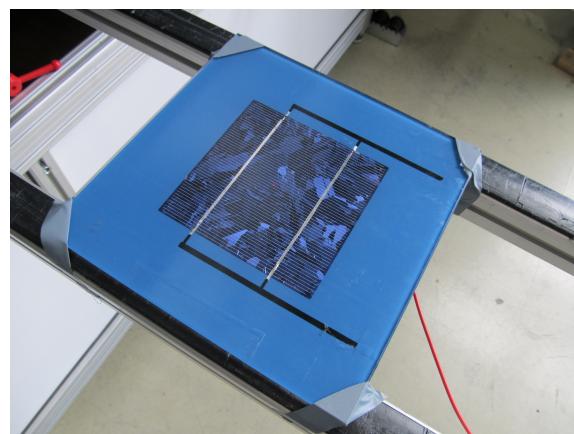


Abbildung 3.1.: Messzelle auf der Lade des Flashers

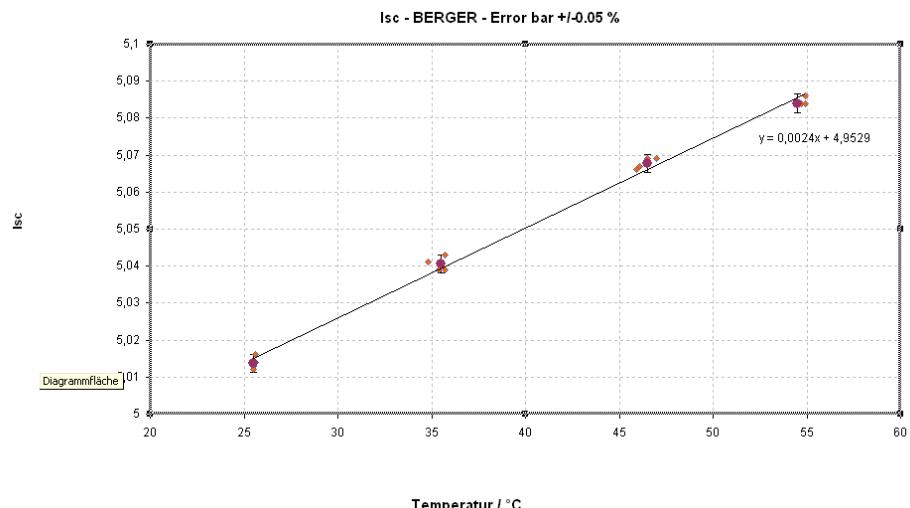


Abbildung 3.2.: Kurzschlusstrom über Temperatur

### **3.3. Strommessung**

### **3.4. Thermische Stabilität**

# **4. Messung**

## **4.1. Messaufbau**

### **4.1.1. Platten**

### **4.1.2. Ablauf**

## **4.2. Auswertung**

wie von sd zu Bild, gemessene Verteilungen,

## **4.3. Schlussfolgerung**

- Code: arduino, Matlab, fileformat

# **Abbildungsverzeichnis**

2.1.	Mecanum-Rad . . . . .	2
2.2.	Rolle . . . . .	3
2.3.	Foto eines gedruckten Rades . . . . .	3
2.4.	Chassis . . . . .	5
2.5.	.....	6
2.6.	.....	6
2.7.	Sensorarray . . . . .	6
3.1.	Messzelle auf der Lade des Flashers . . . . .	8
3.2.	Kurzschlusstrom über Temperatur . . . . .	8

# **Tabellenverzeichnis**

# **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 fÃ¼r 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 optischen 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;

// fÃ¼r 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 fÃ¼r Messpunkt, Spalte und Reihe
int count = 0;
int count2 = 0;
int row = 1;

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

// notwendig fÃ¼r die Kommunikation mit SD Karte
void setup() {
    pinMode(A10, OUTPUT);
    pinMode(53, OUTPUT);
    SD.begin(chipSelect);
}

// Definiert alle m"oglichen Bewegungen
void zuruck(int sped) //speed ist besetzt!
{
    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) //speed ist besetzt!
{
    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) //speed ist besetzt!
{
    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) //speed ist besetzt!
{
    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() //speed ist besetzt!
{
    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() //speed ist besetzt!
{
    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() //speed ist besetzt!
{
    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 zurÃ¼ckfahren
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();
}

// zurÃ¼ck in den Bewegungsmodus
if(richtung=='v') mode='v';
if(richtung=='z') mode='z';

}
break;

case 'v': // VorwÃ¤rtsfahren
{
    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': // RÃ¼ckwÃ¤rtsfahren  
{  
  
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);
```