Lehrstuhl für Raumfahrttechnik
Prof. Prof. h.c. Dr. Dr. h.c.
Ulrich Walter

Bachelorarbeit

Entwurf und Implementierung eines einheitlichen Steuerkonzeptes für einen Sonnen- und Erdalbedosimulator im RACOON-Labor

RT-BA 2018-14

Autor:
Daniel Faust

Betreuer: Dipl.-Ing. Martin Dziura
Lehrstuhl für Raumfahrttechnik
Technische Universität München

Danksagungen

Ich möchte an dieser Stelle besonders meinem Betreuer, Herrn Dipl.-Ing. Martin Dziura danken, welcher mir ermöglichte die Arbeit nach eigenen Vorstellungen zu gestalten und Fragen jederzeit konstruktiv unterstützt hat. Außerdem gilt mein Dank den Herren Thibault Bitenc und Johannes Hilfer für ihren Rat und ihre Hilfe bei der Umsetzung der Arbeit, sowie meinen Kollegen aus der Fachschaft Maschinenbau, welche mir bei Problemen jederzeit mit gutem Rat weiterhalfen.

Zusammenfassung

Die Zahl der Satelliten, Raumfahrzeuge und Schrottteilchen, welche sich im All befinden, steigt immer weiter an. Um die Annäherung zweier Satelliten im All zu simulieren und Sensoren hierfür zu entwickeln, betreibt der Lehrstuhl für Raumfahrttechnik der Technischen Universität München das RACOON-Lab. Es handelt sich um einen Hardware-in-the-Loop Simulator, welcher zwei Satellitenmodelle, sowie zwei Leuchtquellen für Sonnenlicht und Erdalbedolicht umfasst. Um das reale Lichtspektrum möglichst genau nachzubilden, wird auf Metallhalogenidlampen zurückgegriffen. Um die Lampen nicht zu beschädigen, ist es notwendig einige Sicherheitsaspekte zu beachten. Der Sonnensimulator darf nicht in heißem Zustand, also direkt nach dem Ausschalten, wieder gezündet werden, sondern muss vorher mehrere Minuten abkühlen.

Diese Arbeit dokumentiert den Entwurf, die Programmierung und die Implementierung einer einheitlichen Mikrocontrollersteuerung für beide Simulatoren, sowie die Entwicklung einer grafischen Benutzeroberfläche zur Ansteuerung der Lampen. Es werden die wichtigsten der Arbeit zugrundeliegenden Konzepte erklärt, mehrere Varianten zentraler Bauteile vorgestellt und die jeweils am Besten geeignete daraus ausgewählt.

Zur Steuerung wird ein NodeMCU Board mit einem esp8266 Mikrocontroller eingesetzt. Dieser bietet den großen Vorteil einer bereits implementierten W-LAN Schnittstelle, sowie passender Arduino Bibliotheken. Um die Temperatur der Lampe zu erfassen und überwachen zu können, wird ein Thermoelement in die Steuerung integriert. Damit der Sonnensimulator nicht heiß wieder gezündet werden kann, wird zusätzlich die Ausschaltzeit mit einer Real Time Clock überwacht. Diese liefert unabhängig von der normalen Stromversorgung die aktuelle Uhrzeit und ermöglicht es im Speicher des Mikrocontrollers die Zeit zu speichern, wann die Lampe ausgeschaltet wurde. Die Programmierung der Steuerung findet in der Arduino-Entwicklungsumgebung statt. Es werden sämtliche Sicherheitsfunktionen in die Mikrocontrollersteuerung implementiert, um die Sicherheit der Steuerung vom Bediencomputer unabhängig zu gestalten.

Die grafische Benutzeroberfläche wird mittels C# und Windows Presentation Foundation erstellt. Alle Bedienelemente einer Steuerung finden sich zusammengefasst auf einer Seite, es kann zwischen den beiden Simulatorsteuerungen per Tab gewechselt werden. Damit der Bediener einen Überblick über den Status der Lampen bekommt, werden Temperatur- und sonstige Zustandsinformationen, sowie Fehler- und Statusmeldungen der Steuerung in der Oberfläche angezeigt.

Die Kommunikation zwischen Mikrocontroller und GUI ist mittels HTTP-Server Konzept realisiert. Der Mikrocontroller fungiert als Server und die Bedienoberfläche als Client.

Abstract

The number of satellites, spacecraft and space debris is continuously increasing. In order to simulate the approach of two satellites in space and to develop sensors for this purpose, the Institute of Astronautics of the Technical University of Munich operates the RACOON-Lab. The lab is a hardware-in-the-loop simulator that comprises two satellite models and two light sources for sunlight and earth-albedo light. In order to reproduce the real light spectrum as accurately as possible, metal halide lamps are used. To avoid damage to the lamps, it is necessary to consider some safety aspects. The sun simulator must not be re-ignited in a hot state, i.e. immediately after switching off, but must be cooled down for several minutes beforehand.

This work documents the design, programming and implementation of a uniform microcontroller control for both simulators, as well as the development of a graphical user interface to control the lamps. The most important concepts on which the work is based are explained, several variants of central components are presented and the most suitable ones are selected.

A NodeMCU board with an esp8266 microcontroller is used for control. This offers the advantage of an already implemented W-LAN interface as well as suitable Arduino libraries. In order to measure and monitor the temperature of the lamp, a thermocouple is integrated into the control. To prevent the sun simulator from being re-ignited hot, the switch-off time is additionally monitored with a real time clock. The real time clock provides the current time independently of the normal power supply and allows to store the switch-off time in the memory of the microcontroller. The programming of the control is performed in the Arduino development environment. All safety functions are implemented in the microcontroller control to make the safety of the control independent of the operating computer.

The graphical user interface is created using C# and Windows Presentation Foundation. All control elements of a control system are combined on one page, it is possible to switch between the two simulator controls by tab. To give the operator an overview of the status of the lamps, temperature and other status information as well as error and status messages from the controller are displayed in the user interface.

The communication between microcontroller and GUI is realized via the HTTP server concept. The microcontroller acts as server and the user interface as client.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Akronyme

CSS Cascading Style Sheets

DALI Digital Addressable Lighting Interface

DMX Digital Multiplex

EEPROM Electrically Erasable Programmable Read-Only Memory

EVG Elektronisches Vorschaltgerät

GND Ground

GPIO General Purpose Input/Output

GUI Graphical User Interface

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

I²C Inter-Integrated Circuit

IDE Integrated Developer Environment

KiB kibibyte (1024 byte)

LDO Low Dropout Regulator

LED Light Emitting Diode

MHL Metallhalogenidlampe

PCB Printed Circuit Board

PWM Pulsweitenmodulation

RACOON-Lab Real-Time Attitude Control and On-Orbit Navigation Laboratory

RAM Random Access Memory

RTC Real Time Clock

SPI Serial Peripheral Interface

URL Uniform Resource Locator

UTC Universal Time Coordinated

W-LAN Wireless Local Area Network

WPF Windows Presentation Foundation

XAML Extensible Application Markup Language

XML Extensible Markup Language

1 Einleitung

1.1 Hintergrund und Motivation

Über 1.700 funktionsfähige Satelliten kreisten im Jahr 2017 um die Erde[1]. Sobald diese vor der Funktionsunfähigkeit stehen, werden sie je nach Orbit entweder kontrolliert in einen Wiedereintrittsorbit gesteuert und verglühen dort oder sie werden auf einen dafür definierten Orbit angehoben, in welchem sie keinen Schaden anrichten können. Die ebenfalls mögliche Vorgehensweise, diese Satelliten zu reparieren, sowie noch funktionsfähige Satelliten zu warten um deren Lebensdauer zu erhöhen, wird bisher nicht praktiziert. Dies ist auf die hohen Kosten, aber auch auf die Komplexität eines solches Vorgehens zurückzuführen. Das Real-Time Attitude Control and On-Orbit Navigation Laboratory (RACOON-Lab) am Lehrstuhl für Raumfahrttechnik der TUM versucht sich dem zweiten Problem anzunehmen, indem es eine Simulationsumgebung für die Nahbereichsoperation zweier Satelliten im All bietet. Der zweite Einsatzzweck für den Simulator stellt die Entfernung von Weltraumschrott aus Bereichen dar, in welchen dieser für Satelliten und Raumfahrzeuge gefährlich werden kann.

1.1.1 RACOON-Labor am lrt

Das RACOON-Lab besteht aus einem Raum mit zwei fernsteuerbaren Satelliten, einem Chaser- und einem Targetsatelliten, sowie zwei Lichtquellen für die Simulation des Sonnen- und Erdalbedolichtes. Dafür sind beide Lichtquellen auf einer ovalen Schienenkonstruktion um die zwei Satelliten geführt und bilden das jeweilige Lichtspektrum möglichst genau nach. Im Simulator sind außerdem mehrere Kameras und Sensoren installiert, welche die Vorgänge aufzeichnen. Der Raum ist mit lichtabsorbierendem Stoff ausgekleidet, um die Umgebung im All möglichst genau nachzubilden. Einen weiteren Teil der Einrichtung stellt das Mission Control Center dar, von welchem aus die simulierten Missionen gesteuert und überwacht werden können[2]. Zur Bedienung der Hardware ist außerdem ein Computerarbeitsplatz vorhanden, welcher über ein Wireless Local Area Network (W-LAN) auf die einzelnen Komponenten zugreifen und diese steuern kann. In Abb. 1–1 ist der Einsatzzweck des RACOON-Lab erkennbar. Es wird die komplette Verbindung vom Bedienenden am Boden zu dem System im All simuliert und so eine realitätsnahe Bediensituation der Satelliten dargestellt.

1.2 Ziel der Arbeit

Zur einfacheren Bedienung soll für den Sonnen- und Erdalbedosimulator des RACOON-Lab eine einheitliche Mikrocontrollersteuerung entworfen, gebaut und programmiert werden. Außerdem soll eine gemeinsame grafische Benutzeroberfläche implementiert werden, welche die bestehende Kommandozeilensteuerung des Erdalbedosimulators

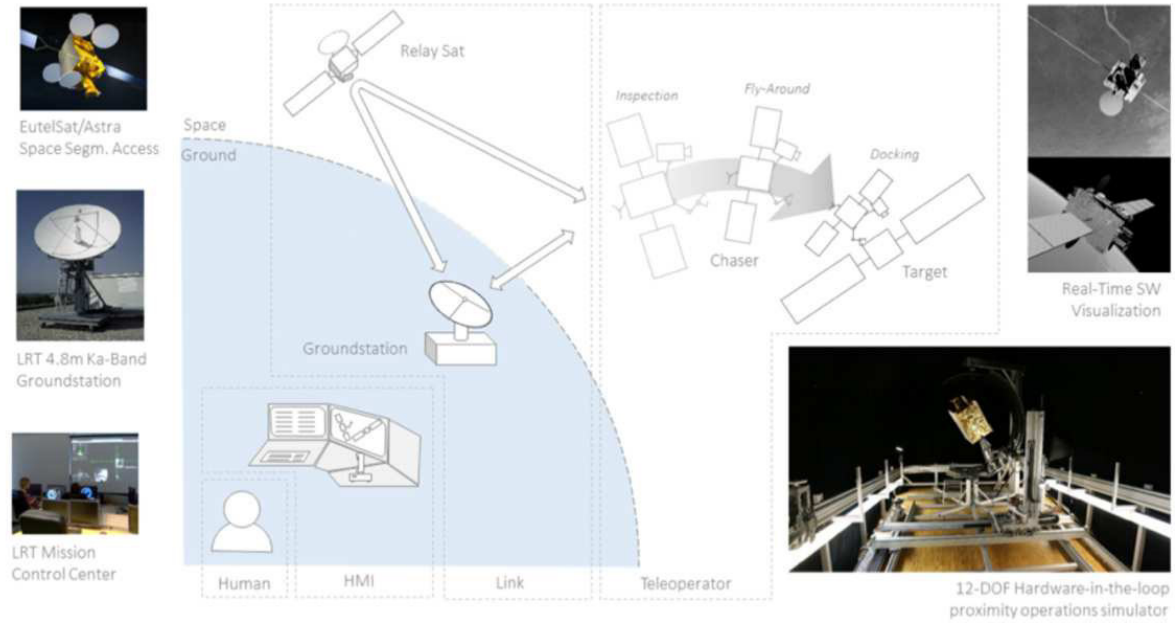


Abb. 1–1: Funktionsprinzip des RACOON-Labors. Der Bedienende kann von einem Kontrollzentrum im Lehrstuhl aus die beiden Satellitenmodelle ansteuern. Hierfür wird der gesamte Simulationsweg vom Kontrollzentrum über eine Bodenstation und optionalem Relaysatelliten zu den Satelliten simuliert[3].

ersetzt. Dabei soll insbesondere der Schutz der Lampen vor zu schnellem Wiedereinschalten sichergestellt und dem Bedienenden der aktuelle Zustand der Simulatoren, in Form von Sensordaten, angezeigt werden.

1.3 Stand der Technik

Bisher besitzt jede der Lichtquellen im RACOON-Lab eine unterschiedliche Steuerung. Beide sind im Rahmen von Studienarbeiten entstanden.

1.3.1 Sonnensimulator des RACOON-Lab

Der Sonnensimulator aus der Bachelorarbeit von Antonio Ciadamidaro[4] besteht aus einer Metallhalogenidlampe (MHL), einem Elektronischen Vorschaltgerät (EVG), welches die Lampe steuert, und einem Zündgerät, das die nötige Spannung zum Zünden aufbauen kann. Das EVG ist nötig, da MHL nur mit hohen Spannungen und Strömen gezündet werden können, welche während des Anschaltvorgangs genauestens gesteuert werden müssen. Des Weiteren ist die MHL nicht in heißem Zustand zündbar, woraus die Notwendigkeit entsteht, die Lampe nach dem Ausschalten zuerst abkühlen zu lassen, bevor sie wieder gezündet werden kann. Um die Lampe auch für kurze Zeit ausschalten zu können, wurde deshalb ein Shuttersystem vor dem Leuchtmittel installiert, welches über einen Servomotor geöffnet und geschlossen wird. Eine genaue

Erläuterung der Funktionsweise von MHL, EVG und Zündgerät findet sich in der Arbeit von Antonio Ciadamidaro[4], hier werden nur die für diese Arbeit relevanten Aspekte aufgegriffen.

Der Sonnensimulator wird momentan mittels zeitgesteuerten Relais ein- und ausgeschaltet. Dieses kann jedoch aus Sicherheitsgründen erst 30 Minuten nach Beginn der Stromversorgung geschaltet werden. Damit wird sichergestellt, dass sich der Simulator auch nach einem Ausfall der Stromversorgung während des Betriebs erst wieder einschalten lässt, wenn dieser abgekühlt ist. Die damit verbundene Wartezeit behindert das Arbeiten mit dem Simulator und soll nun abgeschafft werden.

1.3.2 Erdalbedosimulator des RACOON-Lab

Ähnlich wie der Sonnensimulator ist auch der Erdalbedosimulator aus der Semesterarbeit von Nico Reichenbach [5] aufgebaut. Hier wird ein Halogenid-Metalldampf-Leuchtmittel verwendet, welches auch im heißen Zustand wieder zündbar ist und anders als die MHL des Sonnensimulators dimmbar ist. Die Steuerung der Lampe erfolgt durch ein dem Sonnensimulator ähnliches EVG, welches bereits ein Zündgerät besitzt. Das EVG wird bisher durch eine in der Semesterarbeit entstandene Mikrocontrollersteuerung angesteuert. Die Bedienung dieser Steuerung funktioniert über eine Kommandozeilenanwendung mit Hilfe einer seriellen oder einer W-LAN-Verbindung. Die W-LAN-Verbindung wird über ein 'ESP-01'-Modul des chinesischen Herstellers espressif realisiert, welches an einen 'Teensy 3.2' Mikrocontroller angeschlossen ist.

In Abb. 1–2 sind die beiden Simulatoren zu sehen. Beide sind an bewegliche Konstruktionen montiert, an welchen außerdem die Steuerung, die EVGs und die Stromversorgung angebracht sind.

1.3.3 Ansteuerung von Lichtsystemen in professionellen Anwendungsgebieten

Zur professionellen Ansteuerung von Lichtsystemen werden, je nach Anwendungsbereich, verschiedenste Steuersysteme verwendet. Meist werden die angeschlossenen Geräte mittels Steuersignal von einer zentralen Stelle aus angesteuert. In der Veranstaltungstechnik wird heute fast ausschließlich das Digital Multiplex (DMX) Protokoll genutzt. Dieses bietet eine kurze Reaktionszeit, welche für schnell wechselnde Lichtszenen benötigt wird, sowie die Möglichkeit, eine große Zahl von Geräten parallel zu steuern. Für Szenearien mit niedrigeren Anforderungen, wie der Heimbeleuchtung oder Lichtsteuerung in Gebäuden, auch Architekturbeleuchtung genannt, gibt es eine Vielzahl zusätzlicher Protokolle. Ein häufig genutzter Standard ist Digital Addressable Lighting Interface (DALI)[6]. Neben diesen kabelgebundenen Systemen existieren inzwischen auch Steuersysteme mit kabelloser Signalübertragung wie ZigBee und Z-Wave. Diese Protokolle werden insbesondere bei Smart Home Anwendungen genutzt und sind dort weit verbreitet.

Abb. 1–2: Sonnen- und Erdalbedosimulator im ursprünglichen Zustand. Beide befinde sich im eingebauten Zustand auf der Schienenkonstruktion. An der Unterseite befinden sich die Antriebseinheiten, mit welchen sich die Simulatoren auf der Schiene verfahren lassen. Direkt unterhalb der Leuchtmittel befinden sich die Gehäuse mit EVG, Zündgerät und Steuerung. Unterhalb dieser befindet sich die Stromversorgung.[3]

1.4 Bewertung bestehender technischer Systeme zur Lampensteuerung

Die in Kapitel 1.3.3 aufgeführten Systeme sind für den hier vorliegenden Fall nicht oder nur bedingt geeignet. DMX ist kabelgebunden und daher unbrauchbar. Des Weiteren ist ein Schutz vor zu schnellem Wiedereinschalten nur schwer einzubauen. DALI ist generell nutzbar, allerdings zu teuer in der Anschaffung, da hierfür viele Standardkomponenten benötigt werden. Zusätzlich ist die Kompatibilität mit der bestehenden Infrastruktur nicht vorhanden. DALI braucht, genauso wie DMX eine physische Verbindung in Form von Leitungen. Eine Steuerung über ZigBee oder Z-Wave wäre generell möglich, allerdings bieten diese Funktechnologien nur begrenzte Reichweiten, welche im vorliegenden Problem nicht mit letzter Sicherheit ausreichen und so einen sicheren Betrieb des Simulators nicht gewährleisten können. Da außerdem bereits eine W-LAN Kommunikation für andere Komponenten des RACOON-Lab implementiert wurde, bietet es sich an, diese auch hier zu nutzen.

Da die Kosten für kommerzielle Systeme insgesamt relativ hoch sind, widmet sich diese Arbeit einer möglichst einfachen, kostengünstigen Lösung, welche auf die in Kapitel 2 aufgeführten Anforderungen zugeschnitten ist.

1.5 Angewandte Arbeitsmethodik

Für diese Arbeit wird der Ansatz des Systems Engineering genutzt, um die Anforderungen an das System systematisch im Voraus festzulegen und unerwartete Probleme zu vermeiden. Die verwendeten Methoden stammen aus der Vorlesung “Systems Engineering“ von Prof. Dr. Ulrich Walter und dem zugehörigen Skript[7]. Zunächst werden im Kapitel 2 die Systemanforderungen definiert und je nach Wichtigkeit in unterschiedliche Kategorien eingeteilt. In Kapitel 3 wird eine Einführung in die der Arbeit zu Grunde liegende Technik vermittelt. Daraufhin wird in Kapitel 4 eine Systemarchitektur entwickelt, welche die Realisierung der einzelnen Anforderungen ermöglicht. Hierfür wird in einem Bewertungsverfahren aus mehreren in Kapitel 3 vorgestellten Ansätzen die geeignetste Lösung ausgewählt. In Kapitel 5 wird überprüft, ob die in Kapitel 2 definierten Anforderungen erfüllt wurden.

2 Anforderungen

Im Rahmen des System Engineerings wird im Folgenden ein Anforderungskatalog für die Steuerung erstellt, welcher auf den Kategorien 'MUST HAVE', 'SHOULD HAVE' und 'COULD HAVE' aufbaut. Ein 'MUST HAVE' stellt hierbei eine Anforderung an das System dar, welche unter allen Umständen verwirklicht werden muss, während ein 'SHOULD HAVE' zwar eine wichtige Anforderung darstellt, die jedoch nicht essentiell für das Projekt ist. Ein 'COULD HAVE' stellt eine Anforderung dar, welche zwar wünschenswert für das System ist, jedoch nicht notwendig.

Die zu Beginn der Arbeit entstandenen Anforderungen werden regelmäßig überprüft und angepasst, falls neue Erkenntnisse dies nahelegen. Die zunächst definierten Anforderungen sind also nicht als final anzusehen, sondern bilden einen Rahmen, in welchem die Steuerung entsteht. Sollte während der Arbeit deutlich werden, dass diese unvollständig sind, können sie jederzeit erweitert werden. In Kapitel 5 wird überprüft, ob die einzelnen Anforderungen erfüllt sind. Dort findet sich auch jeweils eine Testmethode, mit welcher sich die Anforderungen überprüfen lassen.

Die in Tab. 2–1 aufgeführten Anforderungen stellen die im Laufe der Arbeit entstandenen, finalen Anforderungen dar.

Grundsätzlich soll die gesuchte Steuerung die beiden Simulatoren an- und ausschalten können. Die Steuerung soll über einen Mikrocontroller erfolgen, welcher mit dem EVG der jeweiligen Lampe, sowie mehreren Sensoren zur Zustandsüberwachung verbunden wird. Diese Mikrocontrollersteuerung wird über eine Schnittstelle mit dem Bediencomputer des RACOON-Lab verbunden und soll durch diesen gesteuert werden. Die Sensoren sollten hierfür mindestens die Temperatur der Lampe überwachen und übertragen. Zusätzlich sollen mehrere Light Emitting Diode (LED) den Zustand der Steuerung am Simulator anzeigen. In Abbildung 2–1 ist der grundlegende Aufbau des Systems dargestellt.

Tab. 2–1: Anforderungskatalog

Priorität	Nummerierung	Anforderung
MUST HAVE	M.1	Die Leuchtmittel der bestehenden Simulatoren müssen an- und abschaltbar sein
MUST HAVE	M.2	Die Lichtintensität der Leuchtmittel muss einstellbar sein
MUST HAVE	M.3	Der Bedienende muss über die Temperatur der Leuchtmittel informiert werden, sowie darüber ob diese ein- oder ausgeschaltet sind. Des Weiteren soll er über auftretende Fehler in der Steuerung informiert werden
MUST HAVE	M.4	Die Leuchtmittel müssen vor Zerstörung durch zu schnelles Wiedereinschalten geschützt werden
MUST HAVE	M.5	Die Bedienung der Steuerung muss von einem Kontrollarbeitsplatz außerhalb des Simulatorraums aus möglich sein
MUST HAVE	M.6	Die Kompatibilität zu den bestehenden Simulatoren muss gegeben sein
SHOULD HAVE	S.1	Die Steuerung sollte in einer Form für beide Simulatoren geeignet sein
SHOULD HAVE	S.2	Der Status der Steuerung und der Leuchtmittel sollte an den Simulatoren erkennbar sein
SHOULD HAVE	S.3	Der Mikrocontroller sollte mit der Arduinoumgebung kompatibel sein

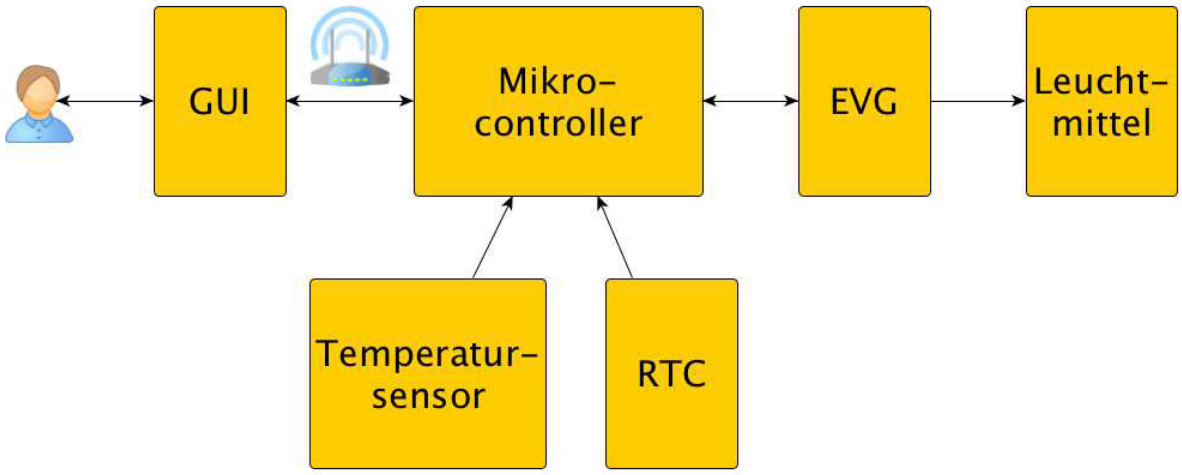


Abb. 2–1: **Grundlegender Aufbau des Gesamtsystems. Der Bediener interagiert mit der grafischen Benutzeroberfläche. Diese kommuniziert drahtlos mit dem Mikrocontroller. Der Mikrocontroller bekommt Daten von einem Temperatursensor, einer Real Time Clock und dem elektronischen Vorschaltgerät der Lampe. Zur Steuerung der Lampe schaltet der Mikrocontroller das EVG.**

3 Grundlagen

In diesem Kapitel findet eine Einführung in die Grundlagen der für diese Arbeit wichtigen Technik statt. Da es außerdem mehrere Ansätze zur Entwicklung der Steuerung gibt, werden verschiedene Bauteile vorgestellt, aus welchen in Kapitel 4.1 die geeignetsten ausgewählt werden.

3.1 Einführung in die Funktionsweise und Programmierung von Mikrocontrollern

Unter einem Mikrocontroller versteht man einen Chip, auf welchem neben einem Mikroprozessor auch ein Speicher und mehrere Ein- und Ausgabeschnittstellen verbaut sind. Ein Mikrocontroller ist darüber hinaus dafür ausgelegt, bestimmte Aufgaben mit möglichst wenigen Bauteilen zu erledigen[8]. Aufgrund der Ausrichtung des Chipaufbaus auf bestimmte Aufgaben gibt es verschiedenste Plattformen und Bauformen.

3.1.1 Arduino

Die Arduinoplattform besteht hauptsächlich aus zwei Teilen, der Arduino-Entwicklungsumgebung, welche auf C und C++ basiert, sowie der Hardware[9]. Ein großer Vorteil der Arduinoplattform liegt in ihrer guten Dokumentation. Ein weiterer Vorteil der Plattform liegt in ihrer Quelloffenheit. Sowohl die Entwicklungsumgebung als auch die Hardware sind 'Open-Source'. Dies bedeutet, dass unter anderem der Code der Bibliotheken und des Bootloaders, sowie die Architektur der offiziellen Hardware frei zugänglich sind. Bibliotheken können beliebig verändert werden und Hersteller können eigene kompatible Boards konstruieren und verkaufen. Programmierer und Hersteller sind unabhängig von einer zentralen Instanz. So entsteht günstige Hardware, welche zusätzliche Funktionen bietet, und entsprechende Bibliotheken, um diese Boards auf die selbe Weise programmieren zu können, wie die offiziellen, durch die Arduino LLC vertriebenen[10]. Die Nachteile solcher Boards sind insbesondere die meist schlechtere Dokumentation und die damit verbundene längere Einarbeitungszeit.

Die Entwicklungsumgebung besteht aus mehreren Teilen. Bei der Sprache, in welcher das Programm implementiert wird, handelt es sich größtenteils um C. Teilweise ist auch C++ wiederzufinden. In den Standardbibliotheken der Entwicklungsumgebung sind die wichtigsten Funktionen zusammengefasst, welche für die Programmierung von Mikrocontrollern benötigt werden. Möchte man zusätzliche Funktionen nutzen, können weitere Bibliotheken eingebunden werden. Ein weiterer Teil der Umgebung stellt der Editor mit integriertem Compiler dar, in welchem der Code geschrieben, kompiliert und auf die Boards geflasht bzw. überspielt werden kann. Nutzer brauchen also nur eine Software, um ihre Mikrocontroller nutzen zu können. In dieser Integrated Developer Environment (IDE) können außerdem Bibliotheken heruntergeladen, sowie Beispiele abge-

rufen werden. Zuletzt gehört auch der Bootloader zur Arduino-Entwicklungsumgebung. Der Bootloader ist ein Programm, welches bei der Herstellung eines Arduino auf den Mikrocontroller aufgespielt wird. Dieses Programm ermöglicht die Programmierung des Mikrocontrollers ohne spezielle 'Programmer'[11]. Programmer werden benötigt, um eigene Programme auf den Mikrocontroller zu übertragen. Der Bootloader vereinfacht den Vorgang, da nun die serielle Schnittstelle eines Mikrocontrollers genutzt werden kann um das Programm zu übertragen. Es wird also keine zusätzliche Hardware benötigt. Der Nachteil eines Bootloaders besteht darin, dass er Speicherplatz auf dem Mikrocontroller belegt, welcher nicht mehr für Programmcode genutzt werden kann[10].

3.1.2 Speicher

Der auf dem Chip integrierte Speicher lässt sich in einen flüchtigen und nichtflüchtigen Teil aufteilen. In dem nichtflüchtigen Speicher werden der Programmcode und Daten gespeichert, welche auch nach Verlust der Stromversorgung erhalten bleiben müssen. Im flüchtigen Teil werden Daten gespeichert, die der Prozessor zum Arbeiten benötigt und die während der Ausführung des Programms entstehen und verändert werden. Dieser deutlich schnellere Random Access Memory (RAM) wird jedoch im Gegensatz zum nichtflüchtigen Speicher bei Ausfall der Stromzufuhr gelöscht und ist daher nicht dafür geeignet, Daten dauerhaft zu speichern[8]. Bei nichtflüchtigen Speichern lassen sich außerdem mehrere Arten unterscheiden, wobei im Folgenden zwei für diese Arbeit wichtige vorgestellt werden. Der Flashspeicher, auch Flash-EEPROM, ist ein wieder beschreibbarer Speicher, welcher für eine große Anzahl an Schreibvorgängen ausgelegt ist. Auf ihm wird meist der Programmcode abgelegt und gelegentlich auch für den Betrieb wichtige Konstanten und Daten. Er lässt sich nur in größeren Blöcken löschen, weshalb stattdessen oft der normale Electrically Erasable Programmable Read-Only Memory (EEPROM) für das Ablegen von einzelnen Daten genutzt wird. Dieser lässt auch das Löschen einzelner Bytes zu, ist jedoch nicht so oft wieder beschreibbar[8]. Der Nachteil von nichtflüchtigem Speicher liegt generell in der langsameren Lese- und Schreibgeschwindigkeit im Vergleich zu flüchtigem Speicher.

3.1.3 Programmierung

Arduino-Programme folgen immer einem einheitlichen Grundaufbau. In Abb. 3–1 ist dieser zu erkennen. Die zwei Funktionen `setup()` und `loop()` sind immer vorhanden. `Setup()` wird einmalig nach dem Start des Programms aufgerufen, während `loop()` danach aufgerufen wird, bis der Mikrocontroller ausgeschaltet wird. Vor dem Aufruf von `setup()` können außerdem noch Präprozessorbefehle programmiert werden.

Arduino Programme werden mit C und C++ geschrieben. Im Beispiel wird zunächst den Variablen `buttonPin` und `ledPin` ein Pin zugeordnet. In der `setup()` Funktion wird jedem Pin eine Funktion zugewiesen. `ledPin` fungiert als Output, `buttonPin` als Input. Während der `loop()` Funktion fragt der Code jedes Mal ab, ob am Eingang `buttonPin`

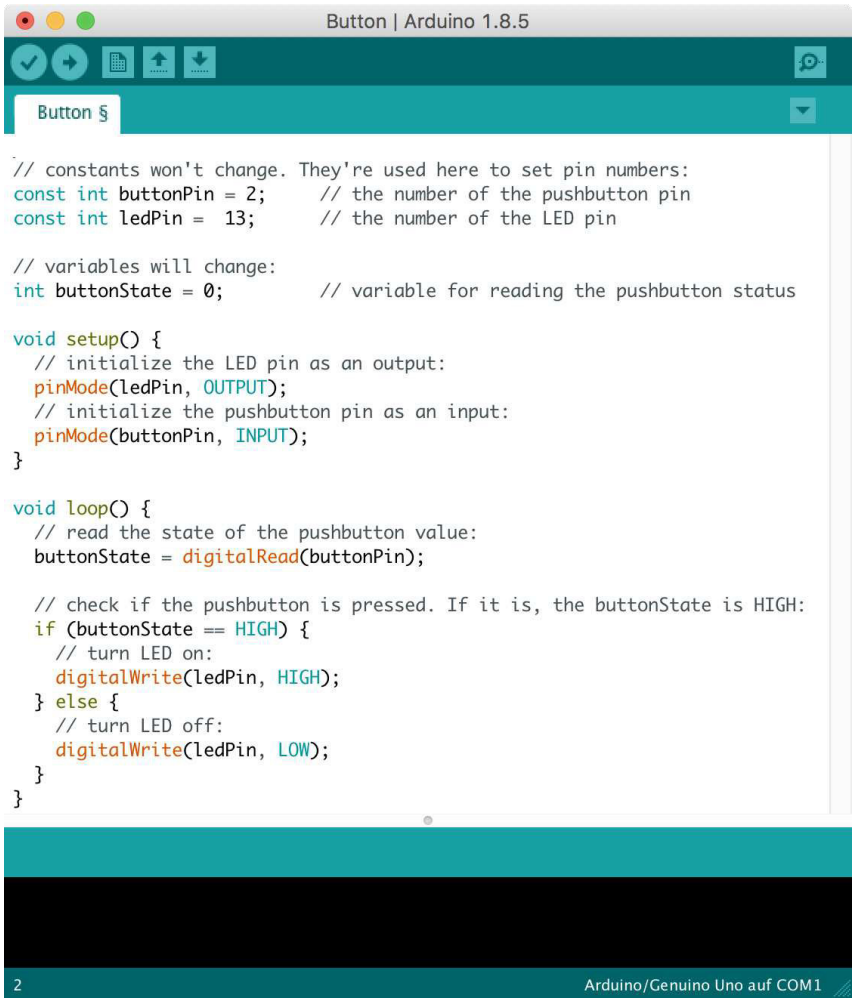


Abb. 3–1: Beispiel für ein Arduinoprogramm, geschrieben in der Arduino IDE. In der setup()-Funktion wird der Modus von zwei zuvor im Präprozessorbereich definierten Pins definiert. In der void()-Funktion wird abgefragt, ob ein Button gedrückt wurde und falls ja, eine LED eingeschaltet.

ein Signal anliegt. Ist dies der Fall wird der `ledPin` auf 'HIGH' gesetzt, ansonsten auf 'LOW'.

3.2 Vorstellung von relevanten Komponenten für die Steuerung

Um die Steuerung zur verwirklichen, können mehrere Ansätze verfolgt werden. In diesem Unterkapitel werden mehrere Bauteile für die Steuerung vorgestellt, aus welchen in Kapitel 4.1 die geeignetsten ausgewählt werden.

3.2.1 Kommunikation zwischen Mikrocontroller und Graphical User Interface (GUI)

Damit Befehle und Daten zwischen dem Bediencomputer und der Mikrocontrollersteuerung ausgetauscht werden können, wird eine Verbindung zwischen diesen benötigt. Diese soll mittels W-LAN hergestellt werden. Zur Realisierung der W-LAN-Verbindung bestehen folgende Möglichkeiten.

Zusätzliches Modul

Wie bereits in der Semesterarbeit von Nico Reichenbach[5] geschehen, kann die Kommunikation zwischen Mikrocontroller und Bediencomputer über ein separates W-LAN-Modul geschehen. Vorteile dieser Lösung stellen insbesondere die Flexibilität bei der Auswahl des Mikrocontrollers und die gute Dokumentation der Lösung dar.

Mikrocontroller mit integriertem W-LAN-Modul

Als Alternative zu einem eigenen Modul besteht die Möglichkeit ein Mikrocontrollerboard mit bereits integrierter drahtloser Kommunikationseinheit zu nutzen. Dies würde den Vorteil bieten, die Anzahl der Bauteile zu reduzieren und eine gut integrierte Kommunikationslösung zu nutzen.

3.2.2 Hypertext Transfer Protocol (HTTP)-Server

Zur Kommunikation zwischen GUI und Mikrocontroller wird eine W-LAN-Verbindung genutzt. Die Kommunikation innerhalb des Netzwerks wiederum funktioniert mittels eines HTTP-Servers. Bei HTTP handelt es sich um ein Kommunikationsprotokoll zwischen zwei Teilnehmern eines Netzwerks. Beim Serverprinzip handelt es sich um eine Art der Kommunikation. Ein Teilnehmer fungiert als 'Server', ein oder mehrere Weitere als 'Client'. Für den Datenaustausch sendet ein Client eine Anfrage, ein 'Request' an den Server, dieser antwortet daraufhin mit einem 'Response'. In der Anfrage sagt der Client dem Server, welche Information er benötigt. Je nachdem, ob der Server diese Information liefern kann oder nicht, antwortet dieser mit der gewünschten Information oder einem Fehler.

3.2.3 Mikrocontroller

Das zentrale Bauteil der Steuerung stellt der Mikrocontroller dar. Da in der Anforderung S.3 aus Kapitel 2 die Kompatibilität mit der Arduinoumgebung gefordert wird, werden nur solche Controller betrachtet.

Da die inoffiziellen, aber zu Arduino kompatiblen Boards meist billiger sind und spezifischere Funktionen bieten, wird für diese Arbeit auf solche zurückgegriffen. Die Vorauswahl fällt auf die beiden Mikrocontrollerboards NodeMCU mit esp8266 Mikrocontroller,

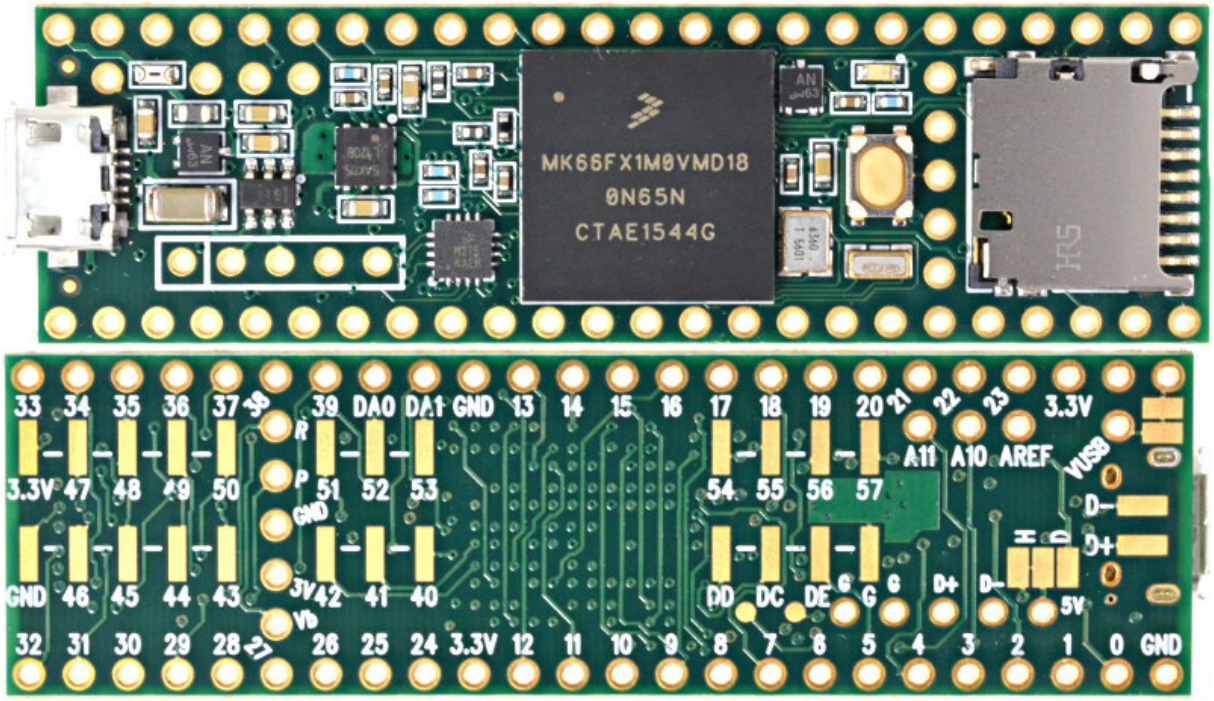


Abb. 3–2: Ober- und Unterseite eines Teensy 3.6[13]

sowie die Teensyfamilie. Teensyboards wurden bereits in den Studienarbeiten von Antonio Ciadamidaro[4] und Nico Reichenbach[5] verwendet, auf welchen diese Arbeit aufbaut. Daher bietet es sich an, diese auch für diese Arbeit erneut zu verwenden. Der esp8266 Mikrocontroller ist ebenfalls bereits am Lehrstuhl bekannt. Es besteht also bereits Erfahrung mit dem Board, auf welche zurückgegriffen werden kann. Außerdem handelt es sich um ein günstiges Board mit bereits integriertem W-LAN-Modul, welches zur Kommunikation zwischen Mikrocontroller und GUI genutzt werden kann. Im Folgenden werden die Vorteile der beiden Boards erörtert. Daraufhin soll in Kapitel 4.1 das für den geplanten Einsatzbereich geeignetste ausgewählt werden.

Teensy

Bei den 'Teensy' Boards handelt es sich um eine gut dokumentierte Mikrocontrollerfamilie von der Firma PJRC. Die Boards besaßen Atmel, jetzt ARM-Prozessoren mit einer 8 bzw. 32 Bit-Architektur. Zusätzlich zum Flashspeicher, auf welchem der Programmcode abgelegt ist, besitzen die Boards noch einen EEPROM mit 1 bis 4 kibibyte (1024 byte) (KiB) Speicher[12]. Vor allem die späteren Versionen sind leistungsstark bei einem geringen Preis und bieten so eine günstige Möglichkeit, Projekte mit Arduino zu erstellen. Mit 25 bis 58 digitalen Input/Output, sowie zwischen 12 und 25 analogen Input Pins ist außerdem eine große Anzahl an Schnittstellen gegeben.

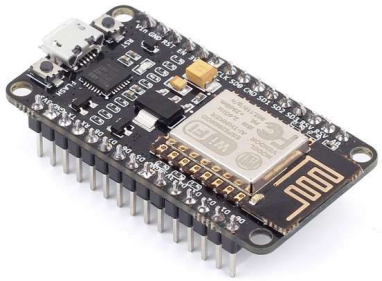


Abb. 3–3: Version 2 eines NodeMCU mit esp8266 von AI-Thinker[14]

Tab. 3–1: Vergleich von Teensy Mikrocontroller und esp8266

Eigenschaft	Teensy 3.6	NodeMCU
Anzahl an GPIOs	58	16
Vorhandener Speicher	1024 kbytes Flash und 4096 byte EEPROM	4 MB
Zusätzliche Funktionen	Bereits integrierte RTC	Eine W-LAN-Schnittstelle ist bereits integriert

esp8266

Bei dem esp8266 des chinesischen Herstellers espressif handelt es sich um einen W-LAN fähigen Mikrocontroller mit einem Tensilica Prozessor[15]. Aufgrund der Möglichkeit, ein Projekt mit dem Chip mit einer W-LAN-Schnittstelle auszurüsten und des günstigen Preises ist er inzwischen sehr beliebt. In Abb. 3–3 ist ein esp8266 auf einem NodeMCU-Board des Herstellers AI-Thinker zu sehen. Am rechten Ende des Boards ist gut der verbaute esp8266 Mikrocontroller mit angrenzender W-LAN-Antenne zu erkennen.

Vergleich

Um eine bessere Übersicht über die Eigenschaften der beiden Mikrocontroller zu bekommen, sind diese in Tab. 3–1 aufgelistet. Aus der Teensyfamilie wird die Version 3.6 verwendet, da diese die aktuellste Version darstellt[12]. Von den Boards, welche einen esp8266 verwenden, wird der NodeMCU des Herstellers 'AI-Thinker' verwendet[14, 16].

3.2.4 Temperatursensor

Im Bereich der Temperatursensoren gibt es unterschiedlichste Typen, welche digitale Daten liefern können. Verbreitet sind beispielsweise Thermoelemente, Widerstandsther-

mometer, Infrarotsensoren und Halbleitersensoren. Sowohl Thermoelemente, Widerstandsthermometer als auch Halbleitersensoren werden zu den Kontaktsensoren gezählt. Darunter versteht man, dass der Sensor fast oder ganz an die zu messende Materie herangeführt wird. Im Gegensatz hierzu existieren die kontaktlosen Sensoren, welche aus einigem Abstand messen. Zu ihnen gehören beispielsweise Infrarotsensoren.

Thermoelemente

Thermoelemente bestehen aus zwei unterschiedlichen Metallen, welche an zwei Stellen elektrisch leitend miteinander verbunden werden. Die erste Verbindung befindet sich an dem zu messenden Punkt und an der zweiten werden die Leiter mit einem Spannungsmesser verbunden. Durch den Seebeck-Effekt baut sich nun eine werkstoffspezifische, elektrische Spannung entlang der Leiter auf. Da es sich um zwei unterschiedliche Metalle handelt, unterscheiden sich die entstanden Spannungen an den Leitern und es kann eine Spannung am Spannungsmesser gemessen werden, welche abhängig von dem Temperaturunterschied zwischen den beiden Verbindungsstellen ist. Über die Temperatur am Spannungsmesser, welche auf anderem Wege ermittelt wird, kann nun die Temperatur an der Messstelle berechnet werden[17]. Damit das Thermoelement auch ausgelesen werden kann, benötigt man außerdem einen Verstärker, welcher die Spannung am Sensor in ein digitales Signal umwandelt. Je nach verwendetem Material gibt es Thermoelemente mit unterschiedlichen Genauigkeiten und abgedeckten Temperaturbereichen. Thermoelemente sind für große Temperaturbereiche geeignet, bieten eine schnelle Reaktionszeit auf Temperaturänderungen und sind leicht zu installieren[18].

Widerstandsthermometer

Widerstandsthermometer bestehen aus einem Metall, dessen Widerstand sich mit der Temperatur ändert. So kann über den auftretenden Widerstand am Sensor auf die Temperatur geschlossen werden. Es werden für gewöhnlich reine, korrosionsbeständige Metalle verwendet, da diese über konstante Temperaturkoeffizienten verfügen. Wird statt einem Metall ein Halbleiter oder Keramik als Sensor verwendet, spricht man von Thermistoren. Vorteile von Widerstandsthermometern liegen in ihrer hohen Genauigkeit und breitem Einsatzbereich. Ein Pt100-Sensor ist beispielsweise, je nach Modell, für Temperaturen zwischen -200 und maximal 850 °C geeignet[19].

Infrarotsensoren

Infrarotsensoren können die ausgestrahlte Infrarotstrahlung von Gegenständen erfassen und wandeln die empfangenen Wellenlängen in den entsprechenden Temperaturwert um. Sie werden unter anderem dann genutzt, wenn ein Sensor nicht ganz an den zu messenden Gegenstand herangeführt werden kann. Da sie etwas Abstand zu dem

Tab. 3–2: Übersicht über die unterschiedlichen Eigenschaften der verschiedenen Temperatursensoren

Eigenschaft	Thermoelement	Widerstandsthermometer
Genutztes Prinzip zur Temperaturmessung	Seebeck-Effekt	Temperaturabhängiges Widerstandsverhalten eines Metalls
Messbare Temperaturspanne in °C	-200 bis 550	-100 bis 500
Genutztes Bus-System	SPI	SPI

Messobjekt benötigen, sind sie für die Unterbringung im Leuchtmittelgehäuse nicht geeignet und werden nicht näher betrachtet.

Halbleitersensoren

Halbleitersensoren messen die Temperatur mittels Dioden oder Transistoren. Deren Schaltspannung ist abhängig von der Temperatur und lässt sich leicht messen. Da der Zusammenhang zwischen Spannung und Temperatur bekannt ist, kann von der Spannung leicht auf die Temperatur geschlossen werden. Halbleitersensoren sind zur Messung von niedrigen Temperaturen geeignet. Der DS18B20 von Maxim Integrated ist beispielsweise für Temperaturen zwischen -80 und 125 °C geeignet[20].

Vergleich der verschiedenen Temperatursensortypen

Wie bereits nach der Vorstellung der Mikrocontroller findet sich in Tab. 3–2 eine Übersicht über die unterschiedlichen Eigenschaften der Sensoren. Die Daten stammen von den zwei Sensoren aus dem Onlineshop `https://www.exp-tech.de`, bei welchem alle Bauteile später bestellt werden.

3.2.5 Real Time Clock

Damit die Steuerung immer die aktuelle Uhrzeit mit dem Zeitpunkt vergleichen kann, zu dem die Simulatoren zuletzt eingeschaltet waren, wird eine RTC benötigt. Hierunter versteht man ein kleines Board mit einem Quarz, dessen Aufgabe es ist, die Zeit auszugeben. Hierfür misst es die Schwingungen des verbauten Quarzes. Damit diese Uhr auch funktioniert, wenn sie einmal nicht an eine externe Stromversorgung angeschlossen ist, besitzen die meisten RTC Boards auch eine eigene Stromversorgung in Form von Batterien oder Akkus[21]. Auf diese Weise kann eine solche Uhr, sobald sie einmal initial eingestellt ist, durchgängig eine Uhrzeit ausgeben. Um die Daten des

Tab. 3–3: Vergleich der beiden RTC Modelle DS1307 und DS3231

Eigenschaft	DS1307	DS3231
Bussystem zur Datenübertragung	I^2C	I^2C
Genauigkeit	Weicht mehrere Sekunden pro Tag ab	Mittels Temperatursensor werden Ungenauigkeiten in der Zeitmessung ausgeglichen
Module mit integrierter Batteriehalterung sind vorhanden	ja	ja

Moduls am Mikrocontroller nutzen zu können, unterstützen RTCs meist ein Bussystem wie Inter-Integrated Circuit (I^2C) oder Serial Peripheral Interface (SPI).

Beispiele für weit verbreitete Module sind das DS1307 und DS3231. Beide sind als Integriertes Modul erhältlich. Oft werden aber auch fertige Module verkauft, auf denen, neben der RTC an sich, zusätzlich elektronische Bauteile zur Spannungsglättung, sowie Batteriehalterungen verbaut sind. Günstigere Modelle, wie das DS1307 gehen nach einiger Zeit meist mehrere Sekunden nach, während besser ausgestattete Produkte versuchen, solche Fehler zu minimieren. Da diese Ungenauigkeiten größtenteils von den ungenauen Quarzen herrühren, deren Schwungfrequenz sich mit der Temperatur minimal ändert, ist in einem hochwertigeren Modul meist ein zusätzlicher Temperatursensor verbaut. Durch diesen kann die RTC temperaturbedingte Abweichungen der Quarzfrequenz mit in die Zeitberechnung einbeziehen.

In Tab. 3–3 ist eine Übersicht über die Eigenschaften der beiden Module zu sehen.

3.2.6 Einführung in Servomotoren und die Datenübertragung mittels Pulsweitenmodulation (PWM)-Signal

Servomotoren gibt es in verschiedenen Ausführungen. Einige bieten nur die Möglichkeit, einen bestimmten Winkelbereich, etwa zwischen $0°$ und $180°$ anzufahren. Andere auch ein Vielfaches von $360°$. Wieder andere haben gar keinen Anschlag und lassen sich beliebig weit drehen. Letztere bezeichnet man auch als 'Segelwindenservos' oder 'continous rotation' Servos. Die Ansteuerung eines Servomotors funktioniert über ein PWM-Signal. Bei einer PWM springt das Signal zwischen den digitalen Werten '0' und '1'. Durch die Dauer, in der das Signal innerhalb einer bestimmten Zeitspanne den Wert '1' besitzt, wird ein Wert übergeben. Außerdem kann mit einer PWM ein analoges Signal mit einem digitalen Outputpin erzeugt werden, indem das digitale Signal schnell zwischen '0' und '1' wechselt.

Im Fall der Servomotoransteuerung wird mittels PWM, je nach Modell, die Position angegeben an welche der Servomotor fahren soll oder, bei Segelwindenservos, die

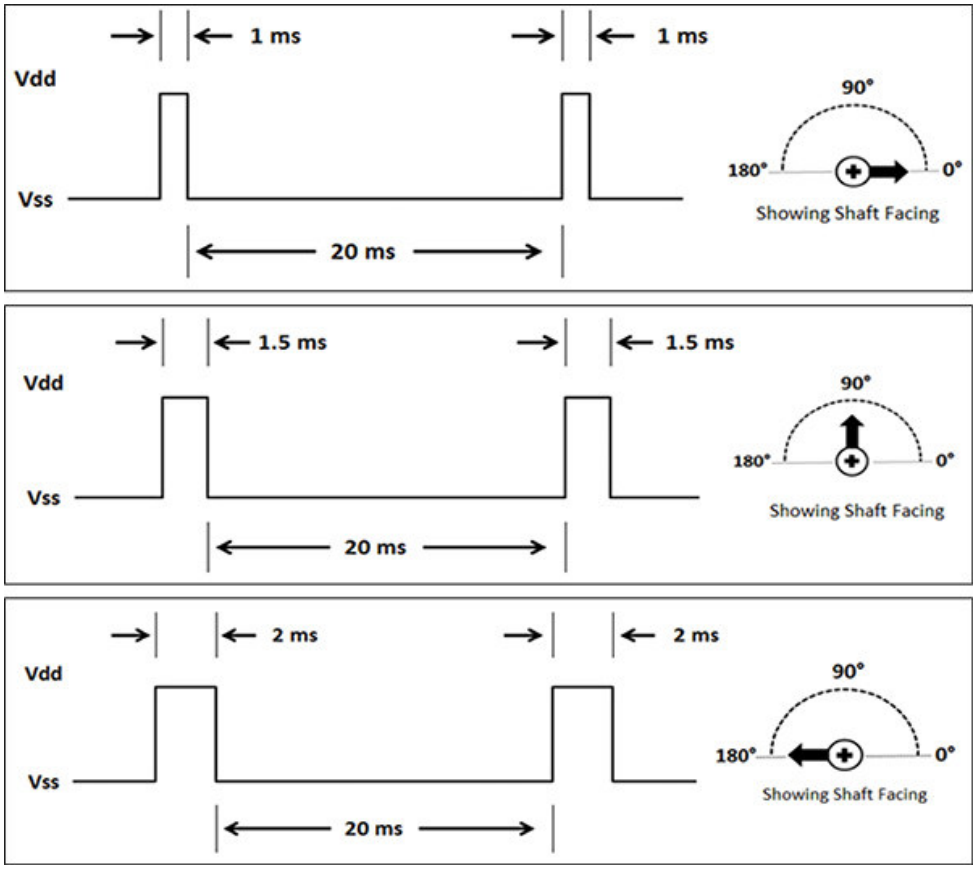


Abb. 3–4: Ansteuerung eines Servomotors mittels PWM-Signal[22]

Richtung, in die er fahren soll, sowie die Winkelgeschwindigkeit. In Abb. 3–4 ist dargestellt, wie dies konkret funktioniert. Ein Zyklus zur Signalübertragung dauert 20 ms, wobei auch Abweichungen von diesem Wert zulässig sind. Je nachdem, wie lange am Anfang des Zyklus das Signal auf ’1’, bzw. ’HIGH’ steht, fährt der Servomotor an eine andere Stelle. Die maximale Zeit für ein ’1’-Signal beträgt beispielsweise 2 ms und die minimale 1 ms. Bei einer Dauer von 1 ms fährt er in diesem Fall an die Position $0°$ und bei 2 ms zu $180°$. Eine Zeitdauer dazwischen führt zu einer entsprechenden Position zwischen $0°$ und $180°$. Bei Segelwindenservos führt eine Dauer von 1,5 ms zu einem Verharren an der aktuellen Position. Will man in die positive Richtung fahren, nutzt man je nach gewünschter Geschwindigkeit einen Wert zwischen 1,5 ms und 2 ms. Unterschiedliche Modelle nutzen meist unterschiedliche Werte für das PWM-Signal. Ein Servo lässt sich dann beispielsweise mit Signalwerten zwischen 0,5 ms und 2,5 ms steuern.

3.3 Programmierung grafischer Benutzeroberflächen

Zum Programmieren grafischer Benutzeroberflächen, auch GUI genannt, gibt es mehrere geeignete Programmiersprachen. Im Folgenden soll kurz vorgestellt werden, inwieweit sich die Programmierung einer GUI vom Entwickeln einer Kommandozeilenan-

wendung unterscheidet und auf die Unterschiede in der Programmierung mit Python, Java und C# kurz eingegangen werden. Daraufhin wird in Kapitel 4.8.4 das für dieses Projekt passendste Verfahren ausgewählt.

Möchte man eine grafische Benutzeroberfläche entwickeln, müssen einige Dinge beachtet werden. Aus der Entwicklung von Kommandozeilenanwendungen ist meist ein Programmaufbau bekannt, in welchem eine Main-Funktion abgearbeitet wird. In dieser wiederum werden einzelne Funktionen der Reihe nach aufgerufen. Es wird oft auf eine Eingabe des Nutzers oder ein anderes Ereignis gewartet und das Programm pausiert so lange. Will man jedoch eine GUI programmieren, ist dieses Vorgehen prinzipiell nicht möglich. Dies liegt an der Art, wie mit einer GUI interagiert wird. Zwar hat auch eine GUI-Anwendung einen Einstiegspunkt mit einer `main()`-Routine, in welcher die Anwendung initialisiert wird, die Interaktion des Nutzers mit dem Programm läuft jedoch anders ab. Während man bei einer Kommandozeilenanwendung nur an einer Stelle Text eingeben und damit eine Steuerung der Anwendung vornehmen kann, sind grafische Benutzeroberflächen meist so konzipiert, dass an mehreren Stellen eine Interaktion erfolgen kann. Meist gibt es mehrere Buttons, Schalter und Schieberegler, mit denen ein Input vorgenommen werden kann. Würde das Programm immer auf eine Eingabe an einer bestimmten Stelle warten, würde das Programm sich regelmäßig aufhängen. Stattdessen besitzt eine GUI eine ereignisorientierte Programmierung. Es werden mehrmals in der Sekunde alle Bedienelemente überprüft und wenn eines davon genutzt wird, wird ein Ereignis ausgelöst. Dieses Ereignis wiederum ruft eine zugehörige Funktion auf, welche beispielsweise eine Textfläche ändert, einen HTTP-Request startet oder das Programm schließt. Ereignisse können neben einem Klick oder eines Tastendrucks auch Zustände wie 'Der Mauszeiger befindet sich über einer bestimmten Fläche' oder 'Die Uhrzeit ist jetzt 18:00 Uhr' sein. Bei der Programmierung einer GUI muss daher darauf geachtet werden, länger andauernde Schleifen zu vermeiden. Während diese in einer Kommandozeilenanwendung sinnvoll sein können, zum Beispiel um einen Countdown anzuzeigen, würden sie in einer GUI Anwendung das regelmäßige Abfragen aller Bedienelemente pausieren und damit auch die Bedienung der GUI zeitweilig unterbinden. Dies macht sich beim Nutzer durch eine nicht mehr reagierende Anwendung bemerkbar.

3.3.1 Python

Bei Python handelt es sich um eine Interpreter-Programmiersprache, der Code wird also während der Ausführung in Maschinencode übersetzt. Das bedeutet, der geschriebene Quellcode wird nicht, wie beispielsweise bei C, zunächst von einem Compiler übersetzt und es wird ein ausführbares Programm erzeugt. Stattdessen wird zur Ausführung der Python-Interpreter benötigt, welcher während der Ausführung des Programms den Quellcode übersetzt und direkt ausführt. Der Vorteil ist, dass keine Compiler benötigt werden und der Code auf unterschiedlichen Betriebssystemen nutzbar ist. Nachteile entstehen durch die Notwendigkeit des Interpreters zur Ausführung. Während C-Programme nach dem Kompilieren auf jedem Windows- PC ausgeführt werden können, muss zum Ausführen eines Python-Programms zunächst ein Python-

Interpreter installiert werden. Während dieser auf Unixbasierten Systemen wie Linux oder Mac OS X meist bereits vorinstalliert ist, muss dies bei Windows vom Nutzer durchgeführt werden.

Python ist, ebenso wie C# und Javascript eine objektorientierte Programmiersprache.

Zur Programmierung von grafischen Benutzeroberflächen besitzt Python mehrere 'Frameworks' oder auch 'Toolkits'. Bei einem GUI-Framework handelt es sich um eine Zusammenstellung von Werkzeugen zur GUI-Entwicklung. Die Python Frameworks bieten unterschiedliche Funktionsumfänge und sind auf unterschiedliche Einsatzszenarien ausgelegt. So gibt es Frameworks, welche für Browseranwendungen ausgelegt sind, Frameworks für einzelne Betriebssysteme, aber auch solche für mehrere Betriebssysteme. Eine Übersicht ist auf der Python Website vorhanden[23]. Bekannte Beispiele für Python Frameworks sind 'Tkinter' und 'PyQt'. 'Tkinter' ist das Standardframework von Python für GUI Anwendungen und ein Teil von Python, basiert jedoch auf 'Tk' von 'ActiveState'[24]. Bei PyQt handelt es sich um ein für Python angepasstes Framework basierend auf Qt[23]. Da Tkinter das Standardframework von Python ist, wird dieses kurz genauer vorgestellt.

Das Layout der GUI wird in Tkinter über Programmcode erzeugt. Man beschreibt also im Quellcode der Anwendung welche Elemente in das Programm eingefügt werden, wo diese platziert sein sollen und welche Funktion sie besitzen. Dies ist insbesondere für Anfänger schwerer vorzustellen, da nicht direkt ein Abbild der programmierten Oberfläche vorhanden ist.

In Abb. 3–5 ist ein Beispielprogramm zu sehen, in welchem mittels Python Code die unterhalb des Codes zu sehende grafische Benutzeroberfläche erstellt wird.

3.3.2 C# mit WPF

Eine von Microsoft angebotene Möglichkeit, grafische Benutzeroberflächen zu programmieren, stellt Windows Presentation Foundation (WPF) dar. WPF ermöglicht, die Entwicklung von sichtbarer Oberfläche und Quellcode weitestgehend getrennt zu betrachten. Dafür nutzt es die Darstellungssprache Extensible Application Markup Language (XAML) und die Programmiersprache C#. Mit der auf Extensible Markup Language (XML) basierenden Sprache XAML kann die Benutzeroberfläche beschrieben und mit C# der Quellcode der Anwendung geschrieben werden. Die Trennung beider ermöglicht die getrennte Entwicklung, beispielsweise die Entwicklung der Oberfläche durch Designer und die Entwicklung des Programms an sich durch Softwareentwickler. Als Alternative zum getrennten Entwickeln lässt sich die grafische Oberfläche auch durch Code beschreiben[25].

Nutzt man entsprechende Entwicklungsumgebungen wie Visual Studio werden die Oberflächen sowohl als XAML, als auch als Grafik angezeigt, welche darstellt wie das fertige Programm aussehen wird. Die Grafik lässt sich in einem integrierten Designer auch bearbeiten. Elemente wie Buttons, Textbausteine oder Schieberegler lassen sich so auch mit der Maus auswählen und platzieren.

A Simple Hello World Program

```
import tkinter as tk

class Application(tk.Frame):
    def __init__(self, master=None):
        super().__init__(master)
        self.pack()
        self.create_widgets()

    def create_widgets(self):
        self.hi_there = tk.Button(self)
        self.hi_there["text"] = "Hello World\n(click me)"
        self.hi_there["command"] = self.say_hi
        self.hi_there.pack(side="top")

        self.quit = tk.Button(self, text="QUIT", fg="red",
                              command=root.destroy)
        self.quit.pack(side="bottom")

    def say_hi(self):
        print("hi there, everyone!")

root = tk.Tk()
app = Application(master=root)
app.mainloop()
```

Abb. 3–5: Beispiel einer mit Tkinter in Python geschriebenen GUI - Oben der Code und unterhalb das auf Mac OS X ausgegebene Fenster

Tab. 3–4: Übersicht über die einzelnen GUI-Frameworks

Eigenschaft	WPF(C#)	Tkinter(Python)	Javascript
Implementierungsform	Compiler	Interpreter	Interpreter
Designer vorhanden?	ja	ja, je nach verwendetem Framework	ja
Einfaches übertragen auf andere Computer?	ja Es muss nur eine Datei kopiert werden	nein Interpreter muss installiert werden	ja Es ist nur ein aktueller Browser zur Ausführung nötig

Die fertigen Anwendungen werden compiliert und liegen anschließend als ausführbare .exe Datei vor. So kann das Programm einfach auf andere Windows-Computer kopiert und dort ohne weitere Software genutzt werden.

3.3.3 Javascript

Bei Javascript handelt es sich, wie bei Python auch, um eine Skriptsprache. Sie wird also mittels Interpreter ausgeführt. Da Javascript jedoch hauptsächlich für Webanwendungen konzipiert ist, werden mit ihr geschriebene Anwendungen meist im Browser ausgeführt. Zur Ausführung von in Javascript geschriebenen Programmen ist daher in der Regel nur ein Browser notwendig.

Zum Einstieg in die GUI Programmierung ist JavaScript weniger geeignet, da zum Einstieg in die Sprache bereits Kenntnisse mit Hypertext Markup Language (HTML) und Cascading Style Sheets (CSS) notwendig sind[26].

3.3.4 Übersicht

In Tab. 3–4 ist eine abschließende Übersicht der drei vorgestellten Programmiersprachen mit ihren GUI-Frameworks gegeben.

4 Systemarchitektur

Nachdem in Kapitel 3.2 mehrere Mikrocontroller und Module vorgestellt wurden, findet in Kapitel 4.1 ein Vergleich dieser statt und es werden mit Hilfe der Anforderungen aus Kapitel 2 die geeignetsten ausgewählt.

Außerdem werden weitere für die Konstruktion der Steuerung wichtige Teile vorgestellt. Im Anschluss wird in Kapitel 4.6 ein Schaltplan und in Kapitel 4.7 ein Platinendesign entwickelt. In Kapitel 4.8 schließlich, wird zunächst die Programmierung des Mikrocontrollers und anschließend die der grafischen Benutzeroberfläche durchgeführt. Hierzu wird außerdem eine Methode zur Programmierung von GUIs aus den in Kapitel 3.3 vorgestellten ausgewählt.

4.1 Auswahl der Hauptkomponenten der Steuerung

4.1.1 Mikrocontroller

In Tab. 4–1 wird mittels gewichteter Punktebewertung der am Besten geeignete Mikrocontroller aus den in Kapitel 3.2.3 vorgestellten ausgewählt.

Es werden 5 Punkte für eine vollständig erfüllte Anforderung vergeben. Wird eine Anforderung überhaupt nicht erfüllt, erhält der Mikrocontroller 0 Punkte. Teilweise erfüllte Anforderung werden mit einer Punktzahl dazwischen bewertet, je nachdem, inwieweit die Mikrocontroller diese erfüllen. Die Gewichtung der einzelnen Anforderungen orientiert sich am Anforderungskatalog aus Kapitel 2. Eine Anforderung, welche erfüllt werden muss, um ein 'MUST HAVE' zu erfüllen, wird mit dem Faktor 3 gewichtet. Erfüllt eine Anforderung ein 'SHOULD HAVE' oder vereinfacht die Realisierung eines 'MUST HAVES', erhält die Anforderung die Gewichtung 2. Ansonsten wird die Anforderung einfach gewichtet.

Nach dem Vergleich der beiden Boards und Auswertung aller Punkte sticht der NodeMCU mit einem esp8266 in der 2. Generation heraus. Durch die native W-LAN-Unterstützung muss kein zusätzliches Bauteil zu Kommunikation auf dem Board implementiert werden und die Anzahl der Schnittstellen zwischen Mikrocontroller und sonstigen Bauteilen auf der Steuerplatine reduziert sich. Durch die geringere Gesamtkomplexität ist auch die Fehleranfälligkeit geringer. Die Ein- und Ausgänge sind ausreichend und die beiden benötigten Bus-Systeme I^2C und SPI werden unterstützt. Die schlechtere Dokumentation kann durch bereits vorhandene Erfahrung mit dem Mikrocontroller am Lehrstuhl kompensiert werden.

4.1.2 Temperatursensor

Im Folgenden wird ein Temperatursensor aus den in Kapitel 3.2.4 vorstellten Typen ausgewählt. Hierfür werden in Tab. 4–2 die Anforderungen an den Sensor aufgelistet

Tab. 4–1: Auswahl eines Mikrocontrollers

Anforderung	Gewichtung	NodeMCU v2 (esp8266)	Teensy 3.6
Ausreichende Anzahl an Ein- und Ausgängen verfügbar (10 werden benötigt)	3	5	5
Kompatibilität zu SPI und I^2C	3	5	5
Schnittstelle zur drahtlosen Kommunikation	2	5	0
Gute Dokumentation	1	2	5
Summe	-	**42**	**35**

Tab. 4–2: Auswahl eines Temperatursensors

Anforderung	Gewichtung	Thermoelement	Widerstandsthermometer
Kompatibilität zu SPI oder I^2C	3	5	5
Schnelle Reaktionszeit	2	5	5
Messbereich bis 500 °C	3	5	5
Summe	-	**40**	**40**

und mit dem aus Kapitel 4.1.1 bekannten Bewertungsschema bewertet, wie gut die unterschiedlichen Sensoren diese erfüllen. Als Bewertungsobjekt für das Thermoelement wird ein Typ-K-Sensor aufgeführt, bei den Wiederstandsensoren wird ein Pt100-Sensor bewertet. Beide sind im Onlineshop `https://www.exp-tech.de` erhältlich, wo nach der Auswahl alle Submodule der Steuerung gekauft werden. Die Sensoren müssen mindestens bis 500 °C messen können, da in ersten Messungen beim Betrieb der Lampe nicht mehr als 150 °C erreicht wurden. Da bei Problemen mit dem Leuchtmittel die Temperatur jedoch deutlich höher liegen kann, wird ein größerer Wert gewählt.

Nach Auswertung der Tabelle wird klar, dass beide Sensoren gleichermaßen geeignet sind. Die Wahl fällt auf das Thermoelement, da dieser Sensortyp bereits in der Bachelorarbeit von Antonio Ciadamidaro[4] genutzt wurde und gut funktionierte. Typ K bezeichnet die Bauart des Sensors, hier eine Paarung aus den Legierungen Nickel-Chrom und Nickel-Aluminium. Der messbare Temperaturbereich liegt zwischen -200°C und 500°C[27]. Zum Auslesen wird ein Adafruit-Board mit einem MAX31855-Verstärker genutzt. Dieser gibt die Temperatur über eine SPI-Schnittstelle aus. In Abb. 4–1 ist das Thermoelement sowie der Verstärker zu sehen. Das Thermoelement besitzt eine Glasfaserhülle, welche den hohen Temperaturen widersteht, sowie eine Edelstahlspit-

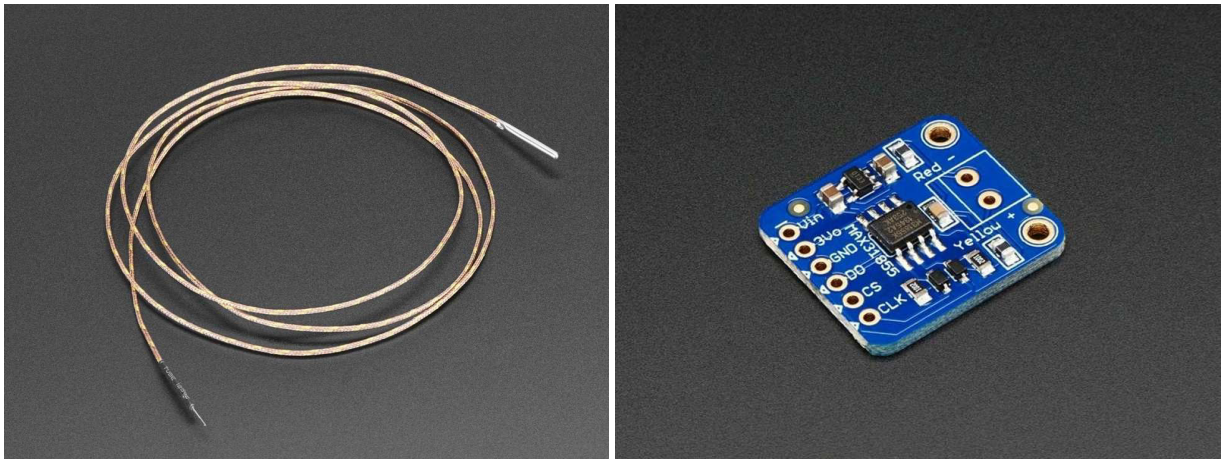


Abb. 4–1: Ansicht des Thermoelements Typ K (links)[27] und des MAX31855 Verstärkers (rechts)[28]

Tab. 4–3: Auswahl eines RTC-Moduls

Eigenschaft	Gewichtung	DS1307	DS3231
Ausreichende Genauigkeit bei der Zeitmessung	1	5	5
Kompatibilität zu SPI oder I^2C	3	5	5
Es existieren Module mit alternativer Stromversorgung (Batterie)	2	5	5
Summe	-	**35**	**35**

ze zum Schutz des Sensors. Der Anschluss an den Verstärker ist über zwei farbig markierte Drähte am Ende des Sensors realisiert.

4.1.3 Real Time Clock

In Tab. 4–3 werden die verschiedenen Eigenschaften der RTCs DS1307 und DS3231 aufgelistet und inwieweit diese wichtig für die Steuerung sind. Auch hier wird das aus Tab. 4–1 bekannte Auswahlverfahren angewandt.

Werden alle Eigenschaften der beiden RTCs verglichen, fällt auf, dass beide RTC gleichermaßen geeignet sind. Da die DS3231 nur etwas teurer als die DS1307 ist, dafür jedoch genauere Zeitmessung bietet, wird erstere RTC verwendet.

Es kommt eine DS3231 als Adafruitmodul zum Einsatz. Dieses bietet eine Batteriehalterung, einen Spannungswandler von 5 V auf 3,3 V, sowie spannungsglättende Bauteile. In Abb. 4–2 ist das Modul von beiden Seiten zu sehen. Auf der Oberseite ist das DS3231-Modul erkennbar, auf der Unterseite das Batteriemodul.

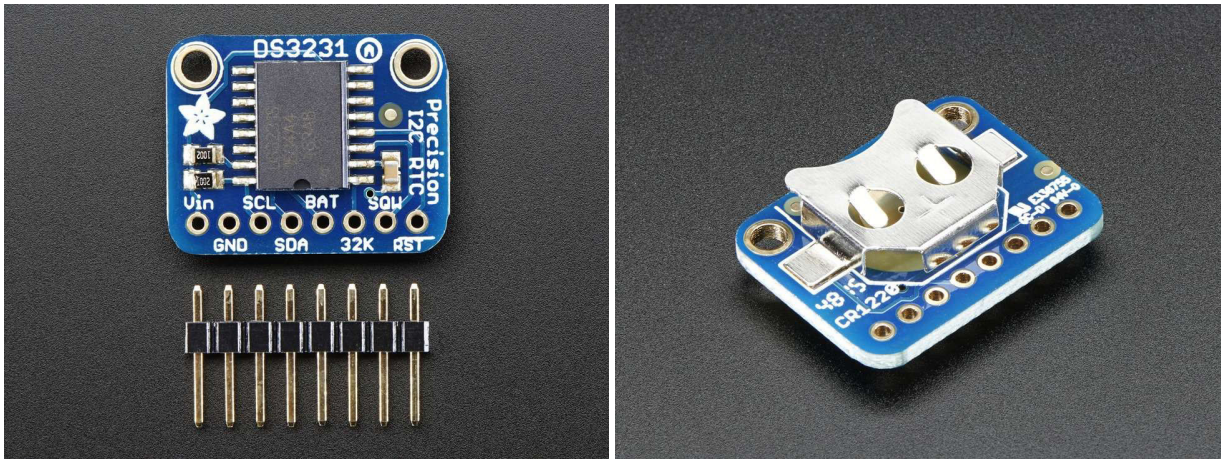


Abb. 4–2: Ansichten des Adafruit DS3231 Moduls von oben (links) und unten (rechts)[29]

4.2 Statusanzeige

Um den momentanen Status des Boards auch direkt an der Steuerung erkennen zu können, werden am Gehäuse der Platine zwei LEDs angebracht. Eine zeigt im Betrieb den Status der W-LAN-Verbindung an. So kann bei einem Fehler schnell die Drahtlosverbindung als Quelle ausgeschlossen werden. Die zweite LED zeigt den Status der Lampe an. An ihr ist leicht zu erkennen, ob die Lampe momentan abkühlt, zündbar ist oder angeschaltet ist. Außerdem gibt sie mittels Blinken wieder, wenn Fehler in der Steuerung auftreten. Auf beiden LEDs wird im Bootvorgang die Konfiguration der Steuerung mittels gemeinsamem Blinken wiedergegeben. So ist leicht zu erkennen, ob der Mikrocontroller zur Steuerung des Sonnen- oder Erdalbedosimulators vorgesehen ist und Beschädigungen durch fehlerhafte Installation sind leicht vermeidbar. Die genaue Bedeutung der verschiedenen LED Anzeigen ist in Tab. 4–4 dargestellt.

4.3 Shutter

Zur Bewegung des Shutters des Sonnensimulators wird ein Servo genutzt, welcher noch aus der Bachelorarbeit von Antonio Ciadamidaro stammt und auch für diese Arbeit weiterverwendet wird. Da zu dem verwendeten Servo kein Datenblatt vorliegt, sondern nur einzelne Daten aus dem Onlineshop von Conrad[30] bekannt sind, wird auf ein Datenblatt für einen Servo zurückgegriffen, welcher der Produktbezeichnung annähernd gleicht[31]. Dieser besitzt die selben Abmaße und stimmt mit den bekannten Daten für den verwendeten Servo vollständig überein. Nach Tests erweisen sich auch die restlichen Angaben des Datenblatts als korrekt, weshalb dieses als Quelle herangezogen wird. Es handelt sich um einen Servo, welcher sich um ein Vielfaches von $360°$ drehen kann, in diesem Fall sind $1800°$ möglich. Die Signalwerte zur Ansteuerung liegen zwischen 800 ms für die Position $0°$ und 2200 ms für $1800°$. Zur Ansteuerung kann die esp8266-Bibliothek `Servo.h` verwendet werden, die auf der gleichna-

Tab. 4–4: Bedeutung der verschiedenen LED Anzeigen. Zunächst das Verhalten der LEDs im Bootvorgang. Nur hier blinken die LEDs simultan. Danach übermitteln sie Informationen zu unterschiedlichen Subsystemen und sind separat zu interpretieren. Bei LED 2 wird Blinken mit höherer Priorität als konstantes Leuchten behandelt.

LED 1	LED 2	Bedeutung
3 Mal weißes Blinken für 1 Sekunde (tritt nur während des Bootvorgangs auf)		Der Erdalbedosimulator ist in der Steuerung eingestellt
3 Mal blaues Blinken für 1 Sekunde (tritt nur während des Bootvorgangs auf)		Der Sonnensimulator ist in der Steuerung eingestellt
Gelbes Leuchten	-	Es besteht momentan keine W-LAN Verbindung. Es wird versucht eine Verbindung herzustellen
Grünes Leuchten	-	Eine W-LAN Verbindung besteht
-	Grünes Leuchten	Die Lampe wird eingeschaltet oder ist bereits eingeschaltet
-	Gelbes Leuchten	Die Lampe kann eingeschaltet werden
-	Rotes Leuchten	Die Lampe kühlt ab und kann nicht eingeschaltet werden
-	Rotes Blinken	Die RTC funktioniert nicht richtig. Ein kritischer Fehler liegt vor und die Lampe kann nicht eingeschaltet werden.

migen Arduino Bibliothek aufbaut. Da der Servo Positionen genau anfährt, kann nun auch der Sonnensimulator mit einer Dimmfunktion ausgestattet werden.

4.4 EVG Kontrollterminal

Damit später ein Schaltplan entworfen werden kann, muss der Aufbau und das Verhalten der EVG-Schnittstelle bekannt sein. Da sich beide EVGs sehr ähneln, findet dies beispielhaft am EVG des Sonnensimulators statt. In Abb. 4–3 ist die Schnittstelle des EVG der Sonne zu sehen, welche mit dem Mikrocontroller angesteuert werden kann. Bei allen Anschlüssen handelt es sich um Optokoppler, welche vom Mikrocontroller geschaltet werden können. Laut Datenblatt des EVG des Sonnensimulators benötigen die Optokoppler dafür eine Spannung von 12 V[32]. Durch Messungen am Kontrollterminal lässt sich jedoch ermitteln, dass auch eine Spannung von 5 V mit 0,7 mA ausreicht, damit die Optokoppler schalten. Hierfür werden die Eingänge 4 und 5 des Kontrollterminals mit den Ausgängen eines 5 V-Netzteils verbunden. Da sich die MHL ohne Probleme stabil anschalten und betreiben lässt, kann davon ausgegangen werden, dass auch 5 V als Schaltspannung ausreichend sind. Durch Messung mit dem Multimeter lässt sich herausfinden, dass der verbaute Optokoppler des EVG bei 5 V Spannung 0,7 A Stromstärke benötigt, um zu schalten. Das EVG des Erdalbedosimulators lässt sich laut Datenblatt mit 5 V ansteuern[33]. In Tab. 4–5 ist aufgelistet, welche Aufgaben die unterschiedlichen Pins an den Kontrollterminals der beiden EVGs erfüllen. Die Funktion kann aus den jeweiligen Datenblättern[32, 33] abgelesen werden.

Bei beiden EVGs ist ein isoliertes und ein nicht isoliertes Kontrollterminal vorhanden, wobei in beiden Fällen auf das isolierte zurückgegriffen wird. Es handelt sich um J101 am Sonnensimulator und J102 am Erdalbedosimulator.

Die Dimmfunktion kann nur am Erdalbedosimulator genutzt werden, funktioniert aber an beiden EVGs nach dem gleichen Prinzip. Um das angeschlossene Leuchtmittel mit voller Leuchtkraft zu betreiben, muss eine Spannung kleiner 0,8 V anliegen. Liegt eine Spannung von 5 V an, wird der Simulator auf 50% seiner Leuchtkraft gedimmt. Dies stellt den maximalen Dimmwert dar. Für alle Werte dazwischen wird ein PWM-Signal mit 100-200 Hz benötigt. Je nach Länge des digitalen ’1’ Werts innerhalb eines Zyklus wird die Lampe entsprechend mit einem Dimmwert zwischen 50 und 100% betrieben[5]. Zur genaueren Erklärung der Funktionsweise eines PWM-Signals sei auf Kapitel 3.2.6 verwiesen.

Die Boostfunktion des EVGs des Sonnensimulators bietet die Möglichkeit, die Ausgangsleistung des EVGs um einen vorher eingestellten Wert zu erhöhen. Dafür muss am entsprechenden Pin ein Signal angelegt werden[32]. Diese Funktion bietet nur das EVG der Sonne, wird jedoch nicht verwendet.

Nun wird geklärt, ob die GPIO-Pins des NodeMCU in der Lage sind, die Eingänge der EVGs sicher zu schalten. Wie bereits erläutert, lassen sich beide EVGs mit 5 V Spannung ansteuern. Da die GPIOs des NodeMCU jedoch nur 3,3 V Ausgangsspannung liefern können, wird nun noch eine Schaltung benötigt, welche diese 3,3 V auf

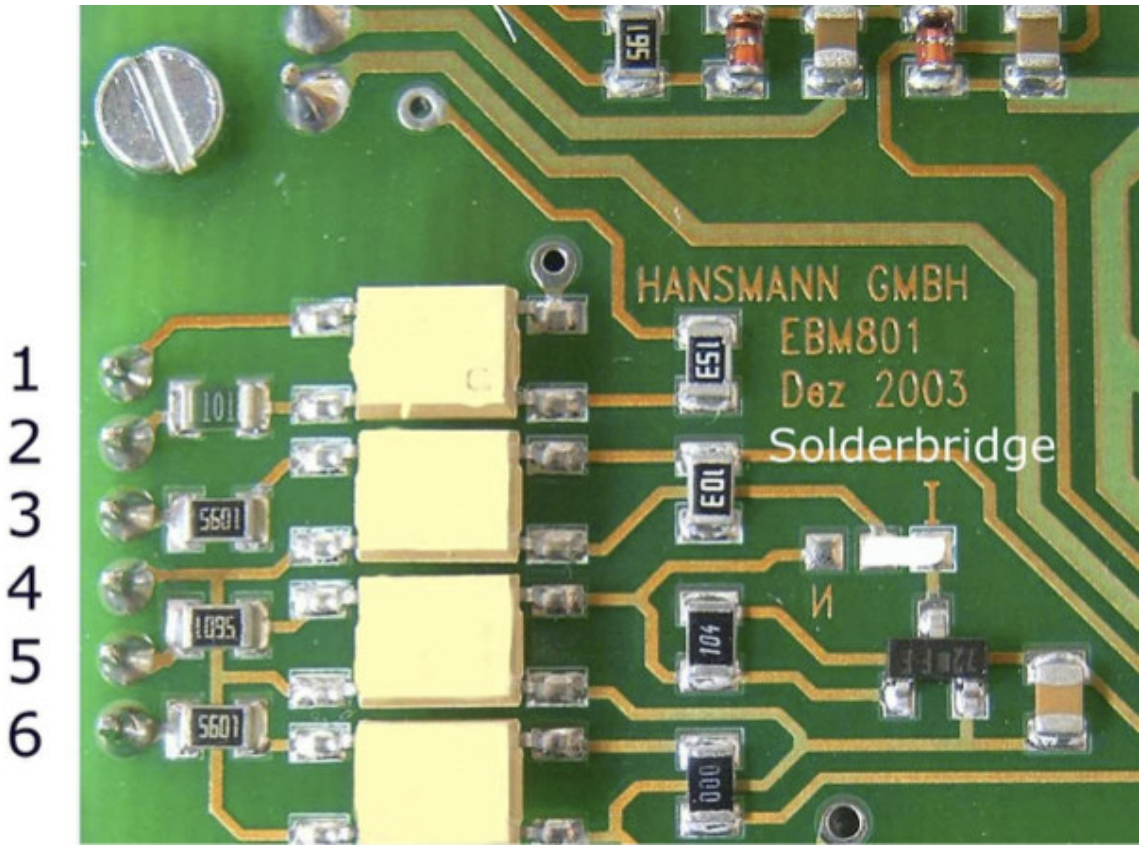


Abb. 4–3: Kontrollterminal J101 des Sonnen-EVG[32]. Links die Nummerierung der Anschlusspins. Mittig die vier Optokoppler zum Ansteuern der verschiedenen Funktionen und Statusübertragung vom EVG aus.

Tab. 4–5: Funktion der einzelnen Pins am EVG-Kontrollterminal von Sonnen- und Erdalbedosimulator

Funktion	**Pin am Simulator**:		**Anmerkung**
	Sonne	**Erde**	
Feedback des EVG, ob die Lampe eingeschaltet ist	1 und 2	5 und 6	Optokoppler funktioniert hier wie ein Schalter. Wenn die Lampe an ist, schaltet der Optokoppler.
Steuerung der **Boostfunktion**	3	nicht vorhanden	Wird in keinem der beiden Simulatoren genutzt.
Ein- und Ausschalten der Lampe	5	3	
Steuerung der **Dimmfunktion**	6	2	Funktioniert nur am Erdalbedosimulator, da das Leuchtmittel des Sonnensimulators kein Dimmen unterstützt.
Masse	4	4	

die benötigten 5 V Schaltspannung transformieren kann. Dafür bieten sich mehrere elektronische Bauteile an. Neben einem Relais, welches für solche geringen Spannungsunterschiede jedoch aufwendig erscheint, ein Transistor oder ein Optokoppler. Da bereits ein Bauteil mit zwei Optokopplern vorhanden ist, wird dieses genutzt. Es handelt sich um einen ASSR1228 von der Firma Avago. Dieser kann das 3,3 V Steuersignal des NodeMCU in ein 5 V Signal für die EVGs transferieren[34]. Der zweite, zum ersten baugleiche, verbaute Optokoppler wird für die Ansteuerung der Dimmfunktion des Erdalbedosimulators genutzt. Ein Optokoppler besteht aus einer LED und einem Photosensor, welche zusammen in einem nach außen lichtundurchlässigen Gehäuse verbaut sind. In Abb. 4–4c ist die Implementierung zu sehen. Die Eingänge der Optokoppler sind an die LED im Inneren angeschlossen und besitzen daher beide einen $400\,\Omega$ Vorwiderstand. Die Ausgänge der Optokoppler, welche an die Photosensoren angeschlossen sind, schalten, sobald die LED leuchtet[34]. Sie werden direkt an die Spannungsversorung des Boards und den zu schaltenden Pin angeschlossen. Wird ein Signal an den Eingang eines Optokopplers angelegt, schaltet dieser und eine 5 V Spannung liegt am gewünschten Pin an.

4.5 Stromversorgung

Nachdem alle Bauteile der Steuerung bekannt sind, wird im Folgenden die Stromversorgung entworfen. Hierfür sind in Tab. 4–6a zunächst alle verbauten Bauteile aufgelistet. Um die benötigte Leistung zu berechnen, ist außerdem aufgeführt, mit welcher Spannung die einzelnen Bauteile betrieben werden. Diese lassen sich aus dem in Ka-

pitel 4.6 entwickelten Schaltplan ermitteln. Zur Berechnung der Leistungsaufnahme der einzelnen Bauteile fehlt noch die benötigte Stromstärke. Diese wird mit einem Multimeter je nach Bauteil unterschiedlich ermittelt. Da die Bauteile außerdem je nach momentaner Aufgabe unterschiedlich viel Leistung benötigen, werden diese in einem Modus gemessen, in dem sie die meiste Leistung verbrauchen.

- Der NodeMCU wird ohne angeschlossene Geräte getestet, um nur die benötigte Leistung des Mikrocontrollers zu erhalten. Um die maximal benötigte Leistung zu erhalten, läuft während der Messung das entwickelte Programm und der NodeMCU verarbeitet wie im späteren Normalbetrieb Server Requests.
- Die Stromstärke der an den NodeMCU angeschlossenen Geräte, also die RTC, das Thermoelement mit Verstärker und der Optokoppler wird gemessen, während diese an den Mikrocontroller angeschlossen sind und mit diesem kommunizieren. So wird die maximale Leistungsaufnahme während des Betriebs ermittelt.
- Beide LEDs leuchten während der Messung.
- Der Optokoppler wird mit einem Vorwiderstand von $400\,\Omega$ betrieben.

Die Leistung wird schließlich mit der Formel P=U*I berechnet und ist ebenfalls in Tab. 4–6a aufgelistet. Zum Vergleich wird zuletzt noch die Stromstärke am Anschluss der Platine an die Spannungsquelle gemessen und mit der Ausgangsspannung die Gesamtleistungsaufnahme der Platine berechnet. Auch hier wurden die Messwerte zu einem Zeitpunkt aufgenommen, an dem die Steuerung viel Leistung benötigt. Das in dieser Arbeit entwickelte Programm wird ausgeführt und regelmäßig Requests an den Server geschickt. Außerdem werden beide Optokoppler geschaltet und die Werte von RTC und Thermoelement abgefragt. Hier gab es einen Normalwert, welcher zum größten Teil der Messung anlag und einen Maximalwert, welcher regelmäßig für kurze Augenblicke erreicht wurde. Die Werte dieser Messung sind in Tab. 4–6b zu finden. Der geringe Unterschied des Normalwerts geht vermutlich auf Messungenauigkeiten zurück. Die auftretenden Maximalwerte vermutlich auf Abläufe im NodeMCU, welche nicht näher bekannt sind.

Es ist ersichtlich, dass alle verwendeten Bauteile mit 5 V bzw. 3,3 V Versorgungsspannung betrieben werden können. Da alle Bauteile mit 3,3 V Versorgungsspannung vom NodeMCU versorgt werden, reicht zur Stromversorgung des Boards ein industrielles Netzteil mit einem einzelnen Ausgang aus. Die Wahl fällt auf das DR-15-5 der Firma Meanwell, da dieses mit 6 V Ausgangsspannung und 12 W Leistung alle verbauten Geräte versorgen kann[35]. Da die Spannung aus dem Netzteil für die Mikroelektronik noch nicht ausreichend störungsfrei ist, um die Bauteile zuverlässig zu versorgen, muss noch eine Spannungsglättung vorgenommen werden. Hierfür werden ein LDL1117 Low Dropout Regulator (LDO) der Firma STMicroelectronics[36] und Kondensatoren verbaut, welche eine konstante 5 V Spannung am Eingang der Platinenkomponenten gewährleisten. Der Aufbau der Spannungsglättung ist in Abb. 4–4a zu sehen.

Da der Servomotor bei den Tests mit 5 V nur sehr unruhig zu bewegen war (siehe Kapitel 5) und bei den Tests etwa 1 W Leistung verbrauchte, wird er vorerst nicht in der

Tab. 4–6: a) Auflistung der Eingangsspannungen der einzelnen Bauteile, der gemessenen Stromstärke und des daraus berechneten Leistungsbedarfs
b) Ausgangsspannung des Netzteils, gemessene Stromstärke am Netzteilausgang und daraus berechnete Leistung

(a)

Bauteil	**Eingangsspannung**	**Stromstärke**	**Leistung**
NodeMCU v2	5 V	80 mA	400 mW
MAX31855	3,3 V	930 µA	3,069 mW
DS3231	3,3 V	70 µA	0,231 mW
LED	5 V	40 mA	200 mW
Gesamt	-	-	**603,3 mW**

(b)

	Eingangsspannung	**Stromstärke**	**Leistung**
Normalwerte am Netzteilanschluss der Platine	6 V	100 mA	600 mW
Maximalwerte am Netzteilanschluss der Platine	6 V	400 mA	2,4 W

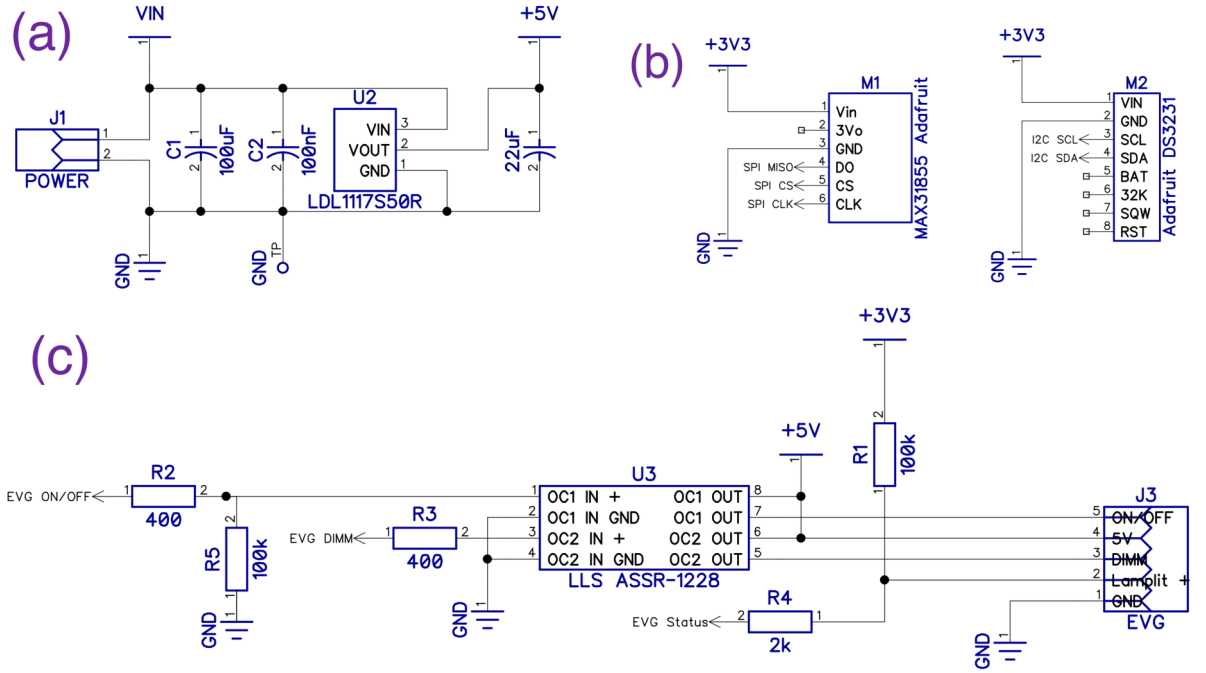


Abb. 4–4: Ausschnitte aus dem Schaltplan der Platine
a) Anschluss des Netzteils an die Platine und Bauteile zur Spannungsglättung
b) Schnittstelle zur RTC und zum Thermoelement
c) Schnittstelle zum EVG mit zwischengeschaltetem Optokoppler

Steuerung verwendet. Dies liegt auch in der Vorsichtsmaßnahme begründet dass der Leistungsverbrauch des Servos zusammen mit den Maximalwerten der Platine den LDO überlasten könnte, was zu Fehlern in der Steuerung des EVGs resultieren kann.

4.6 Entwicklung eines Schaltplans

Nachdem alle Komponenten der Steuerung identifiziert sind, wird ein Schaltplan der Steuerung entworfen. Hierfür wird die Freeware DipTrace verwendet, mit welcher alle nötigen Arbeitsschritte zum Design einer Platine durchgeführt werden können. Mit dem Modul ’Patterns’ der Software lässt sich das Abbild eines Bauteils für die Platine erstellen. Hier werden die Positionen der Anschlüsse des Bauteils genau ausgemessen und in die Software übertragen, damit sich das Bauteil später leicht auf der Platine verlöten lässt. Der ’Component Editor’ ermöglicht das Erstellen von Ersatzschaltbildern für jedes Bauteil, welche schließlich im ’Schematic’ Modul zu einem Schaltplan zusammengesetzt werden können. In jedem Modul sind bereits viele Standardbauteile in der Datenbank vorhanden, so müssen nur noch einige wenige eingetragen werden. Das Modul ’PCB Design’ der Software unterstützt schlussendlich beim Erstellen des Platinendesigns, dies wird im Unterkapitel 4.7 genauer behandelt.

In Abb. 4–4 sind Teile des mit Diptrace erstellten Schaltplans zu sehen. Links oben sind der Anschluss an das Labornetzteil mit den Komponenten zur Spannungsglättung, rechts oben die Module RTC und Thermoelement und unten die Schnittstellen zu EVG und Statusanzeige zu sehen. Der vollständige Schaltplan findet sich in Anhang B.

Um sicherzustellen, dass wichtige Anschlüsse nur schalten wenn dies auch gewünscht ist, wurde außerdem ein Pulldown-Widerstand an die On/Off Steuerleitung angeschlossen. Dieser zieht die Spannung des angeschlossenen Leiters immer auf Ground (GND) herunter. Ohne diese Maßnahme, besteht die Gefahr, dass die Leitung auf einem Wert zwischen digital ’0’ und ’1’ stehen kann, was ein ungewolltes Schalten auslösen könnte. An der Lampenfeedbackleitung, welche den Status der Lampe vom EVG an die Steuerung weitergibt, ist außerdem ein Pullup-Widerstand vorhanden, um das Signal zu invertieren. So kann bei einem Wert ’1’ immer davon ausgegangen werden, dass die Lampe aus ist, während bei ’0’ davon ausgegangen wird, dass diese an ist. Würde man den Wert nicht invertieren, würde man eine defekte Leitung nicht direkt als solche erkennen und eine eingeschaltete Lampe, welche nicht als solche angezeigt wird, birgt größere Gefahren.

Zur besseren Verständlichkeit wurden die Anschlüsse an ’Netzwerke’ angeschlossen. Diese stellen später auf der Platine eine physische Verbindung in Form einer Leiterbahn dar, sind hier aber drahtlos. Des Weiteren wurden für alle Leiter sogenannte ’Testpins’, Abbildung links oben, eingebaut, an welchen die Leiterbahnen leicht mit einem Messgerät angeschlossen werden können. Dies soll später das Testen der Platine erleichtern.

4.7 Platinendesign

Nachdem nun ein Schaltplan vorliegt, kann begonnen werden, diesen in ein Platinenlayout zu überführen. Das dafür vorgesehene Diptrace Programmmodul bietet hierfür die Möglichkeit, ein zuvor entworfenes Schematic zu importieren. Danach werden die Patterns aller verbauten Bauteile angezeigt und es kann angefangen werden, diese anzuordnen. Hierfür werden einige Gestaltungsregeln festgelegt:

- Der Bereich der Platine, über dem später die W-LAN-Antenne des NodeMCU liegt, sollte nach Möglichkeit nicht von Leitern belegt sein, um Störungen bei der Drahtlosverbindung zu verhindern
- Die nach dem Erstellen der Leiterbahnen noch freie Fläche sollte als GND genutzt werden
- Die Ausrichtung des MAX31855 Moduls sollte möglichst so gewählt werden, dass das angeschlossene Thermoelement ohne Probleme aus dem Gehäuse gelegt werden kann. Des Weiteren sollten die Anschlüsse für Schnittstellen leicht erreichbar sein.

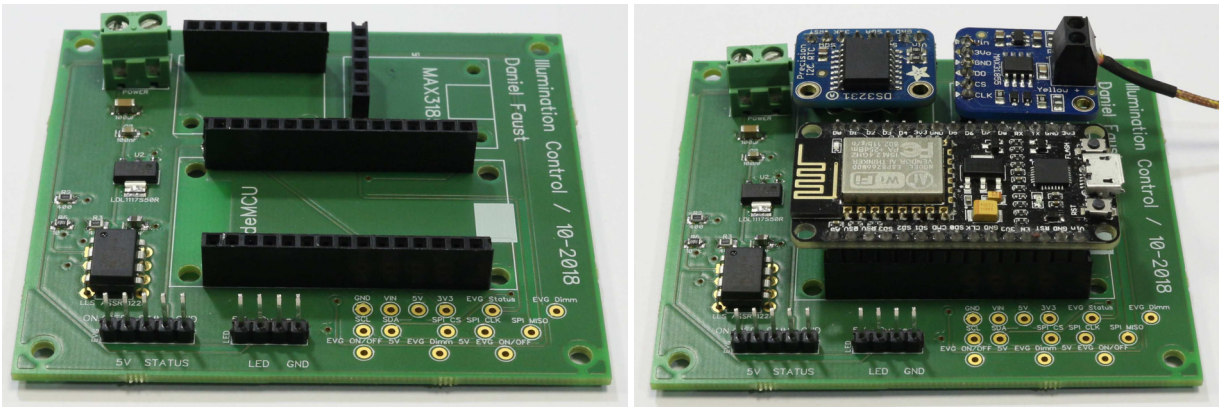


Abb. 4–5: Ansichten des fertigen Boards ohne (links) und mit (rechts) montierten Modulen. Unten links sind die beiden Schnittstellen zum EVG und den LEDs zu sehen. Rechts unten sind Testpins vorhanden um die Platine testen zu können. Auf dem rechten Bild sind zusätzlich das RTC-Modul, das Thermoelement mit Verstärker und der NodeMCU verbaut.

- Insbesondere die Leiterbahnen zu den Bauteilen der Eingangsspannungsglättung sollten möglichst kurz gelegt werden, um den Kondensatoren ein gutes Arbeiten zu ermöglichen

Sind alle Komponenten angeordnet, kann über eine in das Programm implementierte Funktion das Platzieren der Leiterbahnen automatisch vorgenommen werden. Hierfür wird neben der Oberseite mit den Bauteilen auch die Unterseite mitverwendet, welche über Durchkontaktierungen mit der Oberseite verbunden ist. Da es sich um ein vergleichsweise kleines Board handelt, kann auf die Nutzung von zusätzlichen Schichten dazwischen verzichtet werden. Kann Diptrace keine Verbindung aller Pins herstellen, müssen die Bauteile neu angeordnet werden. Sobald alle Leiterbahnen einbaut sind und keine Fehler gemeldet werden, ist das Platinendesign fertig und die Platine, auch Printed Circuit Board (PCB) genannt, kann erstellt werden.

In Abb. 4–5 ist die fertige Platine zu sehen. Die Leiterbahnen sind von einer Kunststoffschicht bedeckt und nicht mehr zu sehen. Unten sind links die Schnittstellen zum EVG und den LEDs zu sehen, rechts Testpins mit welchen das Board getestet werden kann. Sie sind mit den wichtigsten Leiterbahnen verbunden und lassen sich leicht mit einem Multimeter verbinden. Auf dem rechten Bild ist die Platine mit montierten Modulen und NodeMCU zu sehen. Diese müssen nur eingesteckt werden und sind so leicht montierbar. Dies hilft besonders bei Problemen mit einzelnen Modulen, da diese leicht ausgetauscht werden können.

Tab. 4–7: Aufteilung der Funtionen auf GUI und Mikrocontrollersteuerung

Funktionen der GUI	Funktionen der Mikrocontrollersteuerung
• Darstellung der Statuswerte (Temperatur, Lampe an/aus, verbleibende Abkühlzeit, Fehler- und Statusmeldungen) • Möglichkeit, die Lampe an- bzw. abzuschalten • Möglichkeit, den Shutter zu bewegen bzw. die Dimmung der Lampe einzustellen • Anzeige der Debugmeldungen ein- und ausschalten • Übertragen der aktuellen Uhrzeit an die RTC	• Nimmt Anfragen der GUI an • Prüft, wann die Lampe zuletzt an war • Überprüft die Temperatur des Leuchtmittels • Prüft, ob alle Teilsysteme (RTC, Temperatursensor, W-LAN-Verbindung) ordnungsgemäß funktionieren • Verhindert ein Einschalten der Lampe bei einem Fehler in einem Teilsystem und während die Lampe abkühlt

4.8 Programmierung

4.8.1 Anforderungen

Bevor mit der Programmierung begonnen wird, sollen zunächst wichtige Aspekte der Software erfasst werden. Die Wichtigste Anforderung an die Programmierung stellt die Kompatibilität der beiden Softwareprodukte dar. Es muss sichergestellt werden, dass beide über eine geeignete Schnittstelle miteinander ohne Probleme kommunizieren können. Hierfür bietet sich im drahtlosen Bereich das Verwenden eines HTTP-Servers an. In dieser Arbeit stellt die Mikrocontrollerplatine den Server dar, welcher durch eine Anfrage der GUI Befehle zum An-/Ausschalten und weitere Funktionen erhält, während er regelmäßig Statusmeldungen zurückgibt.

Eine weitere Anforderung soll sein, dass die Software des Mikrocontrollers allein einen sicheren Betrieb des Simulators sicherstellen kann. Sie muss also ohne eine Anweisung durch den Nutzer oder die Bediensoftware auf dem Computer bei fehlerhaftem Verhalten den Simulator wieder ausschalten. In Tab. 4–7 ist zu sehen, welche Funktionen der Simulatorsteuerung in der GUI implementiert sind und welche in der Mikrocontrollersteuerung.

Die Software des Mikrocontrollers soll außerdem ohne Anpassung für beide Simulatoren geeignet sein.

4.8.2 Programmierung des Mikrocontrollers

Damit das RTC-Modul und der Digitalwandler des Thermoelements funktionieren, müssen zunächst die nötigen Bibliotheken importiert werden. Für die Simulatorsteuerung werden die durch Adafruit auf der offiziellen Website bereitgestellten Bibliotheken genutzt. Die Bibliothek des RTC-Moduls stellt außerdem eine Definition für das `DateTime`-Format bereit, welche für diese Arbeit ebenfalls genutzt wird. Außerdem wird für das verwendete LED-Modul ebenfalls eine Bibliothek benötigt. Hier kommt die `FastLED` Library zum Einsatz. Damit auch der aus der Bachelorarbeit von Antonio Ciadamidaro[4] stammende Servomotor funktioniert, wird die `Servo` Bibliothek benötigt. Diese ist eine leicht abgewandelte Form der gleichnamigen Arduino-Bibliothek, welche für den NodeMCU angepasst wurde. Weitere durch espressif bereitgestellte Bibliotheken sind die `ESP8266WiFi` und die `ESP8266WiFiMulti` Bibliotheken, welche eine einfache Nutzung des integrierten W-LAN Moduls ermöglichen.

Um den Code für beide Simulatoren gleichermaßen kompatibel zu gestalten, müssen einige Unterschiede in den Code implementiert werden. Deshalb kann zu Beginn mittels eines `enum`-Parameters eingestellt werden, welcher Simulator gesteuert wird. Der Code wird anschließend auf den NodeMCU geflasht und die Einstellung kann nur noch geändert werden, indem der Code geändert und neu geflasht wird. So ist sichergestellt, dass während des Betriebs keine Änderung mehr stattfinden kann. In dieser Einstellung ist gespeichert, wie lange der Simulator abkühlen muss, nachdem er eingeschaltet war. Zusätzlich legt sie fest, ob ein Shutter oder die Dimmfunktion des EVG angesteuert werden kann.

Wie im Programmablaufplan in Abb. 4–6 zu sehen, wird im Setup zunächst per LED ausgegeben, welcher Simulator in der Software eingestellt ist. So lässt sich auch ohne Datenverbindung zum Simulator leicht erkennen, ob die richtige Steuerung angeschlossen ist. Die LEDs blinken dafür beide synchron 3 Mal in derselben Farbe. Näheres zu den Bedeutungen der verschiedenen Farben der LED findet sich in Kapitel 4.2. Nach der Ausgabe des eingestellten Simulators findet noch eine Überprüfung der RTC statt. Arbeitet diese nicht richtig, blinkt die zweite LED rot. Daraufhin wird auch der Status der W-LAN-Verbindung mittels LEDs angezeigt. Fehler, welche durch eine fehlende W-LAN Verbindung oder Probleme mit der RTC verursacht werden, sind so leicht zu erkennen.

Ist eine W-LAN-Verbindung vorhanden und die RTC funktioniert ohne Probleme, erreicht das Programm die `loop()`-Funktion. Dort wird zunächst der Status des EVG abgespeichert. Um Probleme bei der Signalübertragung zu berücksichtigen, wird das Signal dabei invertiert übergeben (siehe Kapitel 4.6) und wird nun wiederum invertiert und abgespeichert. So ist der abgespeicherte Wert ’0’, wenn die Lampe aus ist. Anschließend wird die `Request()`-Funktion aufgerufen, in welcher zunächst überprüft wird, ob eine Anfrage von einem ’Client’ vorliegt. Ist dies der Fall, wird der Befehl analysiert und entsprechende Funktionen zur Ausführung aufgerufen. Ist dies geschehen, gibt die Funktion noch eine Antwort an den ’Client’ zurück und beendet die Verbindung. Die Funktion wird in Kapitel 4.8.2.1 näher beschrieben. Daraufhin überprüft die `Time_check()`-Funktion, ob die Lampe an ist. Ist dies der Fall, wird die aktuelle Zeit al-

le 5 Minuten im EEPROM abgespeichert. Die Beschränkung auf dieses Intervall findet statt, um die begrenzten Schreibzyklen des EEPROM zu berücksichtigen. Zuletzt wird erneut die korrekte Funktionsweise von RTC und W-LAN-Verbindung überprüft. Funktioniert die RTC nicht, werden mittels der `critical_error`-Variable einige Steuerungsfunktionen blockiert. Ein Anschalten der Lampe ist in einem solchen Fall beispielsweise nicht mehr möglich.

Wird die Lampe ausgeschaltet, wird die Zeitspanne, für welche die Lampe eingeschaltet war, zu der bisherigen Laufzeit dazu addiert und im EEPROM abgespeichert. Auf diese Weise kann ungefähr ermittelt werden, wie lange die Lampe bisher lief. So kann besser abgeschätzt werden, wann die Lampe das Ende ihrer Lebensdauer erreicht.

4.8.2.1 Request()

In der `Request()`-Funktion wird nach Auswerten der Anfrage eine Aktion ausgeführt. Die Funktion unterscheidet, wie in Abb. 4–6 zu sehen, zwischen den Anweisungen die Lampe an- oder auszuschalten, die Lampe zu dimmen, beziehungsweise den Shutter zu bewegen, die Statusdaten zu übertragen oder die Einstellungen zu ändern.

Soll die Lampe angeschaltet werden, wird zunächst überprüft, ob ein kritischer Fehler vorhanden ist, beispielsweise, wenn die RTC nicht korrekt funktioniert. Ist dies ausgeschlossen, wird abgefragt, ob seit der letzten An-Phase der Lampe genug Zeit vergangen ist und die Temperatur der Lampe niedriger als 50°C ist. Soll die Lampe ausgeschaltet werden, wird ermittelt, wie lange die Lampe gebrannt hat. Im EEPROM befindet sich die Zeitspanne, die das Leuchtmittel seit der Installation gebrannt hat. Der gerade ermittelte Wert wird dazu addiert und die Summe wieder am selben Ort abgespeichert.

Soll die Lampe gedimmt werden, wird der übertragene Wert mittels PWM an das EVG übertragen und eine Bestätigung mit dem momentanen Dimmwert zurückgegeben.

Soll der Shutter geöffnet oder geschlossen werden, wird die gewünschte Position an den Servo übergeben.

Soll der Speicherort der `last_on` Variable geändert werden, also der Zeit, wann die Lampe zuletzt eingeschaltet war, wird zunächst die aktuell gespeicherte Zeit abgerufen, daraufhin der Speicherort um 7 erhöht und dann die Zeit erneut abgespeichert.

Soll die Zeit der RTC geändert werden, wird die momentane Universal Time Coordinated (UTC) als Unixtime vom Bediencomputer übertragen. Bei der Unixtime handelt es sich um die Zeit in Sekunden, die seit dem 1. Januar 1970 vergangen ist. Der Mikrocontroller wandelt den Wert in das `DateTime` Format um und die Uhr wird damit neu programmiert.

Werden Daten angefordert, werden alle seit der letzten Abfrage angefallenen Status- und Fehlermeldungen, sowie die Temperatur, verbleibende Abkühlzeit, Speicherort der `last_on`-Variable, Uhrzeit, bisherige Brennzeit des Leuchtmittels und der Status der Lampe in einem String zusammengefasst und als Antwort übertragen.

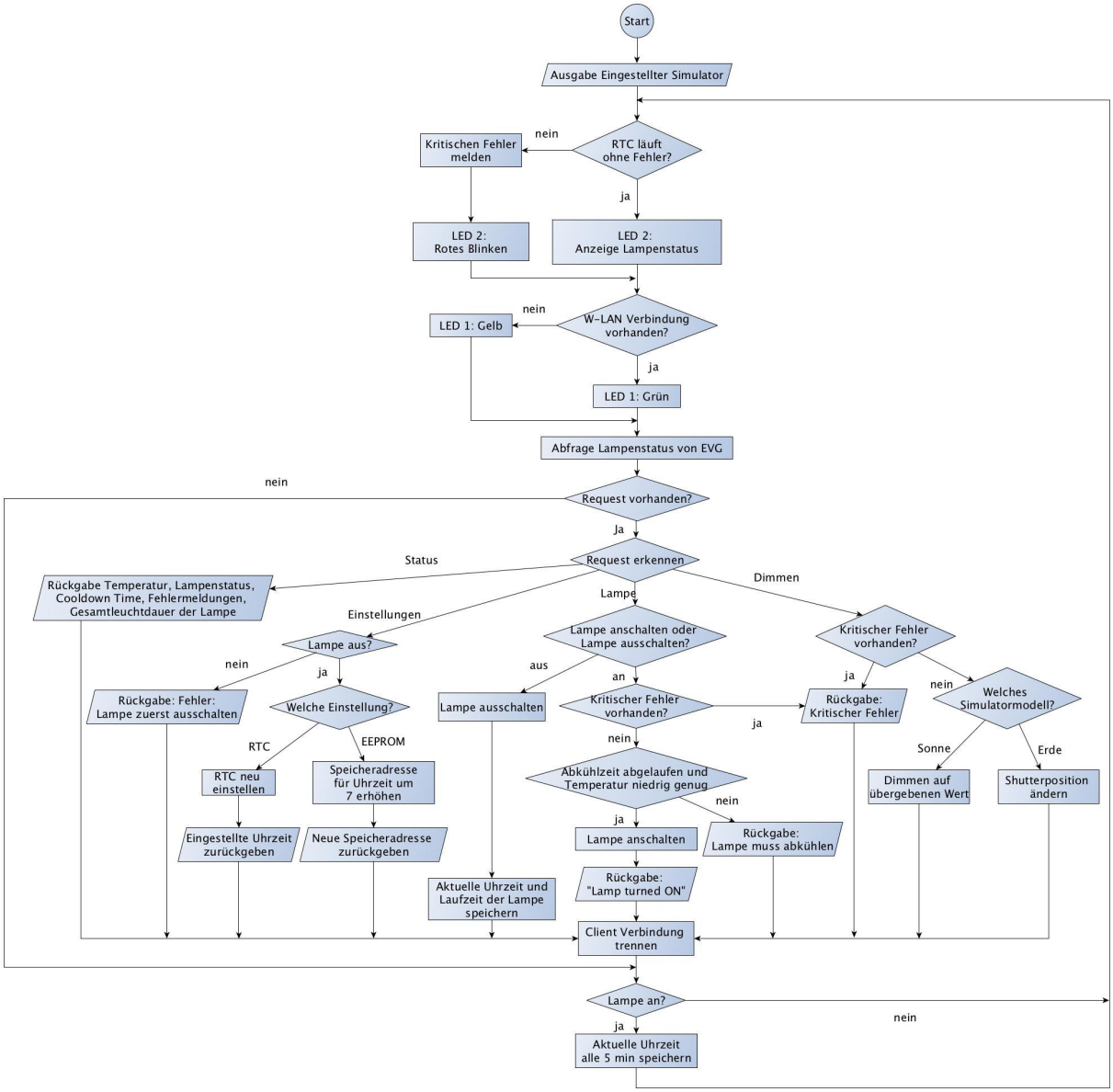


Abb. 4–6: Programmablaufplan der Mikrocontrollersteuerung

Tab. 4–8: Auswahl einer Sprache zur GUI Programmierung

Eigenschaft	Gewichtung	C# (WPF)	Javascript	Python (tkinter)
Für Anfänger geeignet	2	4	3	3
Übertragbarkeit auf andere Computer	2	5	4	2
Es wird keine zusätzliche Software zur Ausführung benötigt	2	5	2	2
Gute Dokumentation	1	4	4	4
Editor zum grafischen Erstellen der GUI vorhanden	2	5	4	3
Gesamt	-	**42**	**30**	**24**

Zuletzt wird in jedem Fall die Antwort auf die initiale Anfrage übertragen und anschließend die Verbindung getrennt.

4.8.3 Auswahl einer Sprache zur GUI Programmierung

Die in Kapitel 3.3 vorgestellten Programmiersprachen besitzen alle unterschiedliche Eigenschaften. In Tab. 4–8 sind diese aufgelistet und bewertet, inwieweit die Eigenschaften auf die einzelnen Sprachen zutreffen.

Die Verwendung von WPF bietet die meisten Vorteile. Durch die Möglichkeit mit Visual Studio die Oberfläche per Drag&Drop zusammenzuziehen und zu gestalten, ist sie als einfacher Einstig in die GUI-Programmierung geeignet. Außerdem lassen sich mit WPF gestaltete Programme leicht auf neuen Computern installieren, da `.exe`-Dateien erzeugt werden, welche nur noch kopiert werden müssen. Da die durch Microsoft bereitgestellte Dokumentation außerdem leicht zu nutzen ist und bereits Erfahrung mit der Sprache am Lehrstuhl besteht, kann Neues leicht erlernt werden.

4.8.4 Programmierung der Grafischen Benutzeroberfläche

Damit die GUI einfach und intuitiv zu bedienen ist, folgt das Design der Oberfläche folgenden Designgrundlagen:

- Meldungen vom Mikrocontroller sollen an einem zentralen Punkt gesammelt und angezeigt werden.

- Wichtige Daten wie die aktuelle Temperatur, verbleibende Abkühlzeit oder der Status der Lampe sollen an einem festen Platz angezeigt und regelmäßig aktualisiert werden.

- Der Aufbau der Steuerungen für die beiden Simulatoren soll in etwa identisch sein.

- Der Zugriff auf die Steuerung soll durch eine Passwortabfrage abgesichert werden.

- Wichtige Informationen sollen zusätzlich zu ihrer Textdarstellung auch visuell dargestellt werden. Insbesondere den Status der Lampe, also ob diese momentan ein- oder ausgeschaltet ist, soll visuell hervorgehoben werden.

- Um Wartezeiten zu vermeiden, soll der Nutzer vor dem Ein- und Ausschalten der Lampe gefragt werden, ob dies wirklich beabsichtigt ist.

Unter Berücksichtigung dieser Grundlagen wird folgender Aufbau der GUI entwickelt. Die GUI besteht aus den drei Fenstern `MainWindow`, `Password` und `Settings`. Innerhalb des Fensters `MainWindow` existieren die beiden Seiten `Sun` und `Earth`. Diese können als Tab angewählt werden und besitzen annähernd den selben Aufbau. In Abb. 4–7 ist das `MainWindow` Fenster mit den beiden Tabs zu sehen. Im rechten Teil der Seiten ist ein Bereich zu sehen, in welchem Statusmeldungen angezeigt werden. Oberhalb kann mittels Checkbox die Debugfunktion aktiviert werden. Ist diese aktiv, werden auch Meldungen angezeigt, welche für den normalen Betrieb unwichtig sind, aber bei Fehlern hilfreich sein können. Oberhalb ist außerdem die aktuelle Temperatur des Simulators ablesbar.

Auf der linken Seite der Oberfläche findet sich ein Button, mit welchem der Simulator an- und ausgeschaltet werden kann. Rechts daneben ist der momentane Status der Lampe farbig dargestellt. Befindet sich der Simulator momentan in der Abkühlphase, wird mittig die verbleibende Zeit zum möglichen Neustart als Countdown dargestellt. Unterhalb kann entweder die Dimmfunktion des Erdalbedosimulators oder der Shutter des Sonnensimulators mittels Schieberegler genutzt werden. Zusätzlich lässt sich der Shutter auch mit einem Button vollständig öffnen oder schließen. Die Bedienoberfläche des Erdalbedosimulators besitzt keinen Button für die Steuerung der Dimmfunktion und sieht daher geringfügig anders aus. Unterhalb der Dimmer- bzw. Shuttersteuerung wird die bisherige Brenndauer des Leuchtmittels angezeigt.

Wird die Anwendung gestartet, wird zunächst das Passwortfenster (siehe Abb. 4–8) angezeigt. Nach Eingabe des korrekten Passworts erscheint das Hauptfenster. Will man Einstellungen an der Speicherstelle der Uhrzeit im EEPROM des NodeMCU vornehmen oder die aktuelle Uhrzeit an den Mikrocontroller übertragen, ist dies im Fenster `Settings` (siehe Abb. 4–8) möglich. Dieses ist durch einen Klick auf die Schaltfläche `Settings` in der linken unteren Ecke des Hauptfensters möglich, wobei auch hier zunächst ein Passwort eingegeben werden muss.

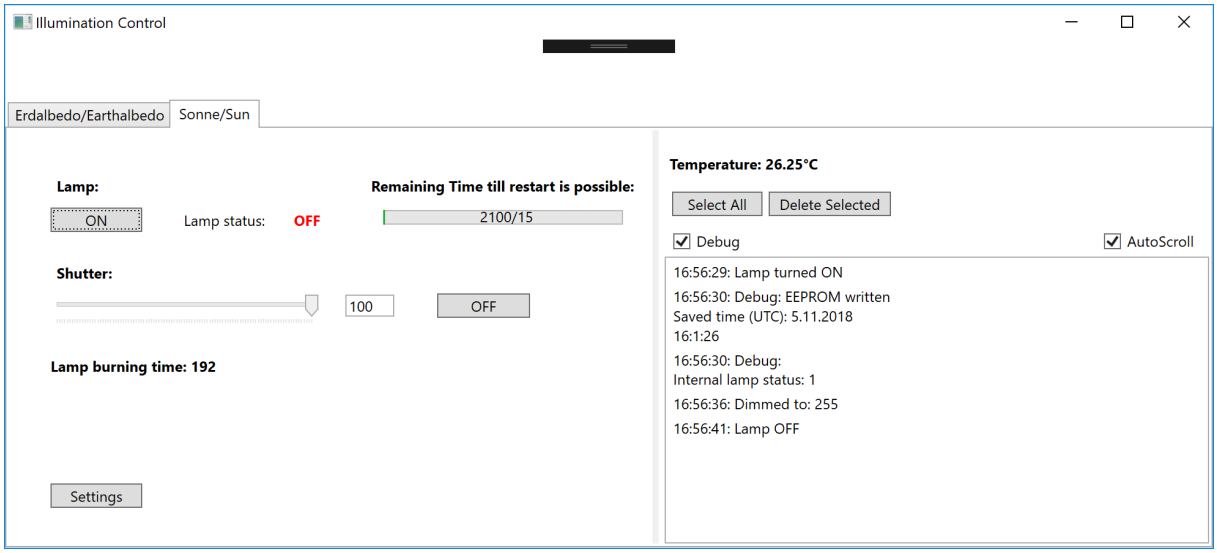


Abb. 4–7: Ansicht des Hauptfensters mit geöffneter Steuerung des Sonnensimulators. Links oben die Auswahl des Simulators per Tab. Darunter Bedienelemente zum An- und Ausschalten, zum Ansteuern des Shutters, sowie die Anzeige der bisherigen Brenndauer des Leuchtmittels. In der Mitte die Darstellung der verbleibenden Abkühlzeit, und rechts die Temperaturanzeige und Status-, sowie Fehlermeldungen mit Zeitstempel.

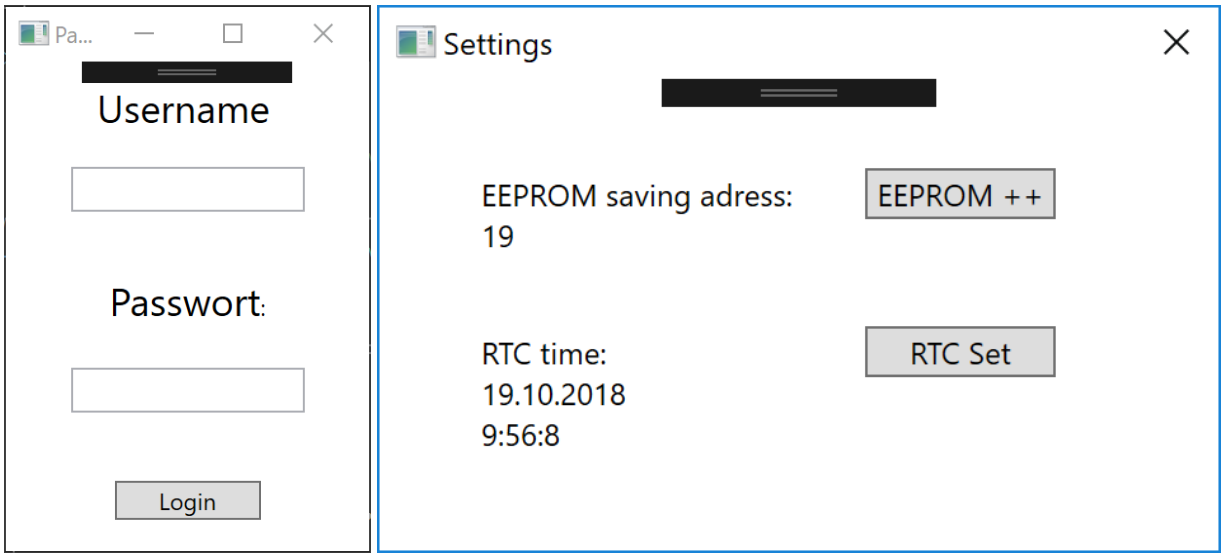


Abb. 4–8: Ansichten der Passworteingabe (links) und des Einstellungsfensters (rechts)

Alle Daten, welche die GUI anzeigt, werden mittels HTTP-Request vom Mikrocontroller geholt. Konkret sind dies:

- Temperatur (Status)
- Lampe an oder Lampe aus? (Status)
- Momentane Speicheradresse der Zeit, wann die Lampe zuletzt an war (Status)
- Aktuelle Zeit der RTC (Status)
- Verbleibende Zeit, bis die Lampe wieder eingeschaltet werden kann (Status)
- Statusmeldungen
- Fehlermeldungen der Steuerung

Bei dem Request wird zunächst von der GUI die Anfrage an den Server, hier der Mikrocontroller gestellt. Je nach Anfrage antwortet dieser mit einer Nachricht. Die mit 'Status' markierten Daten in vorangegangener Aufzählung werden alle durch die selbe Anfrage gesammelt übertragen. Auf diese Weise wird die Anzahl der Anfragen an den Server möglichst gering gehalten. Die Daten werden mittels `&` getrennt und haben unterschiedliche Identifizierer. Mithilfe der Identifizierer, den ersten 7 Zeichen jeder Nachricht, ordnet die Methode `MessageFromLamp_Handling` jede Nachricht einem Zielort zu. Die Temperatur, Abkühlzeit und Lampenstatus werden an entsprechender Stelle angezeigt. Fehlermeldungen und Statusmeldungen werden in einer Liste angezeigt, welche automatisch zum jeweils neuesten Element springt. Die Speicheradresse und aktuelle RTC-Zeit werden im `Settings` Fenster angezeigt. Die Methode `Status_Request`, welche die Statusmeldungen anfordert, wird in einer Schleife alle 500 ms aufgerufen. Damit dies nicht jedes Mal die GUI anhält, wird die Schleife in einem eigenen Thread ausgeführt.

Befehle an den Mikrocontroller werden ebenfalls über eine Anfrage an diesen übermittelt. Der Befehl befindet sich in der Uniform Resource Locator (URL), welche von der GUI aufgerufen wird. Damit das Warten auf die Antwort des Servers nicht die Anwendung blockiert, wird dem Programm mit den Befehlen `async` und `await` signalisiert, dass nicht mit der Ausführung des Codes gewartet werden muss, bis eine Antwort angekommen ist.

5 Verifikation

In diesem Kapitel soll zunächst auf Probleme eingegangen werden, welche während der Entwicklung der Steuerung auftraten. Dafür wird zunächst das aufgetretene Problem und daraufhin die Lösung erläutert. Im Anschluss findet in Kapitel 5.2 eine Überprüfung auf Einhalten der in Kapitel 2 definierten Anforderungen statt.

5.1 Während des Platinendesigns aufgetretene Probleme

Nachdem in Kapitel 4.7 die Platine entworfen wurde, muss diese zunächst mit den Bauteilen bestückt und getestet werden. Nachdem zuerst mit einem Multimeter geprüft wurde, ob alle Leiterbahnen korrekt verbunden sind, wird der erste Prototyp an ein Breadboard angeschlossen um die korrekte Funktion aller Komponenten zu testen. Hier können Fehler noch leicht erkannt und ausgebessert werden. Sind alle erkannten Fehler beseitigt, kann die Steuerung unter Sicherheitsvorkehrungen an die Simulatoren angeschlossen werden.

Im Folgenden sind die Probleme aufgelistet, welche bei Tests auftraten und in der nächsten Version der Platine ausgebessert wurden. Da es sich um kleinere Fehler handelte, konnte das Grunddesign der Platine beibehalten und nur kleinere Teile mussten geändert werden.

Unzureichende Leistung des LDO

Der zunächst verbaute LD1117S50CTR besitzt eine zu große Dropoutspannung. Da das verwendete Labornetzteil nur maximal etwa 5,7 V Ausgangsspannung liefert und der LD1117S50CTR eine Dropoutspannung von mindestens 1 V besitzt, können die zum Betreiben der Bauteile benötigten 5 V nicht bereitgestellt werden[37].

Um das Problem zu lösen, wird ein LDL1117S33R von STMicroelectronics verbaut, welcher eine maximale Dropoutspannung von 600mV besitzt und somit ausreicht[36].

Bootverhalten des NodeMCU

Als die Steuerung erstmals an den Sonnensimulator angeschlossen wurde, trat beim Aktivieren der Stromzufuhr ein kurzes Knacken am Simulator auf und die Stromzufuhr wurde sicherheitshalber unmittelbar deaktiviert.

Die Pins des NodeMCU besitzen beim Bootvorgang unterschiedliche Verhaltensweisen. Der zunächst für das An- und Ausschalten der Simulatoren vorgesehene GPIO 16 besitzt beim Booten einen kurzen Zeitraum den digitalen Wert ’1’ und schaltet so für wenige Millisekunden den Simulator an. Dieses Verhalten konnte durch Anschließen des fraglichen Pins an einen Logic Analyser ermittelt werden. Während der

Überprüfung des Pins wurde außerdem eine Analyse des Bootverhaltens aller GPIO Pins erstellt. Hierbei wurde deutlich dass, mit Ausnahme von GPIO 4 und 5 (im Anhang als D1 und D2 bezeichnet), jeder Pin im Laufe des Bootvorgangs für einige Augenblicke den Wert '1' annimmt. Die Daten des Logic Analysers sind im Anhang A zu finden.

Da die Pins 4 und 5 zunächst zur I^2C Anbindung vorgesehen waren, wurde diese auf die jetzigen Pins verschoben. Daraufhin konnte das An- und Ausschalten von Pin 5 aus erfolgen. Nachdem die neue Konfiguration erfolgreich getestet wurde, wird diese nun final verwendet.

Leistungsbedarf des Servos

Da bei abschließenden Tests ein höherer Leistungsbedarf des Servos als zunächst angenommen festgestellt wurde, und dieser bei der geplanten Spannungsversorgung von 5 V nur unruhig arbeitet, wird dieser vorerst nicht in die Steuerung integriert. Stattdessen wird geplant den Servo parallel zum entwickelten Board an den Ausgang des Netzteils anzuschließen und die Signalleitung des Boards zu verwenden wie geplant.

5.2 Überprüfung der Anforderungen

Im Folgenden wird zu jeder Anforderungen aus Tab. 2–1 ermittelt, ob die Anforderung eingehalten wurde und falls nicht, weshalb dies der Fall ist. Hierfür wird zunächst eine Testmethode definiert, mit welcher die Einhaltung getestet werden kann.

M.1: Die Leuchtmittel der bestehenden Simulatoren müssen an- und abschaltbar sein

Testmethode: In der Steuerung wird die Lampe eingeschaltet. Reagiert die Lampe innerhalb 30 Sekunden, wird der Befehl zum Ausschalten gegeben. Reagiert diese wieder innerhalb von 30 Sekunden und schaltet sich aus, gilt die Anforderung als erfüllt.

Verifikation: Die primäre Funktion der Steuerung, die Anlage an- und auszuschalten, wird erfüllt. Die Funktion ist mittels Klick auf den entsprechenden Button in der GUI erreichbar.

M.2: Die Lichtintensität der Anlage muss einstellbar sein

Testmethode: In der Steuerung wird die Lichtintensität geändert. Ändert sich die Lichtintensität der Lampe innerhalb von 30 Sekunden, gilt die Anforderung als erfüllt.

Verifikation: Ein Einstellen der Lichtintensität wird bei dem Erdalbedosimulator über die in das EVG integrierte Dimmfunktion realisiert. Da die MHL des Sonnensimulators nicht dimmbar ist, ist eine Einstellung der Lichtintensität hier nicht möglich.

M.3: Der Bedienende muss über die Temperatur der Leuchtmittel informiert werden, sowie ob diese ein- oder ausgeschaltet sind. Des Weiteren soll er über auftretende Fehler in der Steuerung informiert werden

Testmethode: Zeigt die Bedienoberfläche innerhalb von einer Minute nach Aktivieren der Stromzufuhr die Temperatur und den Status des Leuchtmittels an, gilt der erste Teil der Anforderung als erfüllt. Daraufhin wird die RTC entfernt und die Verbindung zum EVG getrennt. Zeigt die Steuerung dem Bedienenden entsprechende Fehler an, gilt auch Teil zwei als erfüllt.

Verifikation: Eine Information des Bedienenden über den Zustand der Anlage erfolgt in der GUI. Die aktuelle Temperatur des Leuchtmittels und die Information, ob die Anlage an- oder ausgeschaltet ist, sind immer zu sehen. Kühlt die Anlage ab, sind die verbleibenden Sekunden bis zum möglichen Wiedereinschalten ebenfalls sichtbar. Außerdem werden Fehlermeldungen als Nachrichten angezeigt. Weitere Statusinformationen sind über die implementierte Debugfunktion erreichbar.

M.4: Die Leuchtmittel müssen vor Zerstörung durch zu schnelles Wiedereinschalten geschützt werden

Testmethode: Das Leuchtmittel wird an- und wieder ausgeschaltet. Nun wird alle 30 Sekunden der Befehl zum Anschalten gegeben. Reagiert die Lampe beim Sonnensimulator für 30 Minuten und beim Erdalbedosimulator für 30 Sekunden mit einer Fehlermeldung und die Lampe schaltet sich nicht ein, gilt die Anforderung als bestanden.

Verifikation: Indem eine minimale Abkühlzeit vorgegeben wird und durchgängig die aktuelle Temperatur gemessen wird, ist ein Schutz vor Zerstörung der Anlage durch zu schnelles Wiedereinschalten gewährleistet.

M.5: Die Bedienung der Steuerung muss von einem Kontrollarbeitsplatz außerhalb des Simulatorraums aus möglich sein

Testmethode: Lässt sich die Steuerung von außerhalb des Simulatorraums ansteuern, gilt die Anforderung als erfüllt.

Verifikation: Durch das Platzieren aller Bedienelemente in der GUI ist eine Steuerung von einem beliebigen Computer möglich, welcher sich im selben W-LAN-Netzwerk wie die Steuerung befindet.

M.6: Die Kompatibilität zu den bestehenden Simulatoren muss gegeben sein

Testmethode: Wenn sich die konstruierte Steuerung mit beiden Simulatoren verwenden lässt, gilt die Anforderung als erfüllt.

Verifikation: Die EVGs beider Simulatoren sind mit der selben Platine und GUI möglich. Die Anforderung wurde eingehalten.

S.1: Die Steuerung sollte in einer Form für beide Simulatoren geeignet sein

Testmethode: Wenn sich die konstruierte Steuerung in der selben Form sowohl für den Sonnensimulator als auch für den Erdalbedosimulator nutzen lässt, gilt die Anforderung als erfüllt. Dabei muss die Hardware die Selbe sein, an der Software dürfen kleinere Einstellungen möglich sein.

Verifikation: Da für beide Simulatoren die selbe Platine genutzt werden kann und die grafische Benutzeroberfläche gleichzeitig beide Lampen ansteuern kann, sind die selben Komponenten für beide Simulatoren geeignet. Die Unterscheidung zwischen den beiden Simulatoren findet nur mittels eines Parameters in der Software statt. Die Anforderung ist erfüllt.

S.2: Der Status der Steuerung und der Leuchtmittel sollte an den Simulatoren erkennbar sein

Testmethode: Lässt sich der Status wichtiger Subsysteme der Steuerung direkt an der Steuerungshardware am Simulator ablesen gilt der erste Teil der Anforderung als erfüllt. Lässt sich außerdem erkennen, ob die Lampe momentan abkühlt, eingeschaltet wird oder ist, oder ob die Lampe aus ist, jedoch eingeschaltet werden kann, gilt auch der zweite Teil als erfüllt.

Verifikation: Durch die implementierten LED Leuchten ist jederzeit der momentane Status von grundlegenden Teilen der Steuerung ablesbar.

S.3: Der Mikrocontroller sollte mit der Arduinoumgebung kompatibel sein

Testmethode: Lässt sich der Mikrocontroller mittels Arduino IDE programmieren und lassen sich die Standardbibliotheken verwenden, gilt die Anforderung als erfüllt.

Verifikation: Indem der zur Arduinoumgebung kompatible esp8266-Mikrocontroller genutzt wurde, ist die Anforderung erfüllt.

6 Fazit und Ausblick

6.1 Fazit

Der Hauptanspruch dieser Arbeit, eine einheitliche, intuitive und sichere Steuerung für die beiden Simulatoren zu schaffen, wurde erreicht. Während zuvor beide Leuchtmittel auf unterschiedliche Weise angesteuert wurden und eine unnötige Wartezeit beim Anschalten des Sonnensimulators in Kauf genommen werden musste, ist die Steuerung nun vereinheitlicht und über ein gemeinsames Interface möglich. Der Bediener erhält während des Betriebs durchgängig Statusinformationen und Fehlermeldungen angezeigt. Der Ansatz, alle Sicherheitsfunktionen der Steuerung in den Mikrocontroller zu implementieren, hat sich als erfolgreich erwiesen. Auf diese Weise kann die Ansteuerung einfach in fremde Anwendungen integriert werden. Die grafische Benutzeroberfläche liefert alle relevanten Daten und ist leicht zu bedienen.

Die Nutzung eines NodeMCU zur Steuerung bietet durchaus Vorteile. Die einfache Nutzung der W-LAN-Schnittstelle ist überaus bequem und hilfreich. Die Nachteile, welche mit der Nutzung einhergehen, sind jedoch nicht zu unterschätzen. Besonders das Bootverhalten des Mikrocontrollers birgt Schwierigkeiten für die Nutzung. So scheint im Nachhinein die Anwendung eines anderen Mikrocontrollers durchaus sinnvoll, zur Drahtloskommunikation hätte dann beispielsweise erneut ein 'ESP-01'-Modul als Schnittstelle dienen können, wie bereits in der Semesterarbeit von Nico Reichenbach[5] geschehen. Alternativ wäre ein näheres Betrachten anderer esp8266 basierter Mikrocontrollermodelle empfehlenswert, da viele Probleme auf das NodeMCU-Board zurückzuführen sind und nicht auf den Mikrocontroller an sich.

Das Konzept eine RTC zu nutzen, um die letzte Ausschaltzeit des Leuchtmittels zu speichern ist als erfolgreich anzusehen. Die Simulatoren sind durch das regelmäßige Speichern der Uhrzeit während des Betriebs auch dann geschützt, wenn die Leuchtmittel nicht ordnungsgemäß ausgeschaltet werden und deshalb ein Speichern der Ausschaltzeit nicht erfolgt. Die vorgestellte DS1307 wurde im Auswahlprozess jedoch zu schnell abgelehnt und kann in Zukunft bei ähnlichen Projekten durchaus empfohlen werden, da die Genauigkeit der DS3231 RTC für diese Anwendung nicht nötig ist.

Die Programmierung der GUI in C# mit WPF war intuitiv und der Designer half bei der Erstellung der Bedienoberfläche. Die gute Dokumentation half bei Problemen und das Erstellen der GUI war so auch ohne Vorkenntnisse in der GUI Programmierung gut möglich.

6.2 Ausblick

Um die Ansteuerung beider Simulatoren weiter zu vereinfachen, könnte in Zukunft versucht werden, die Ansteuerung der Lichtquellen in die Steuersoftware des gesamten RACOON-Lab zu integrieren. Da die Ansteuerung mittels URL erfolgt, kann diese auch

einfach in andere Software implementiert werden. Da sämtliche Sicherheitsfeatures in der Mikrocontrollersteuerung vorhanden sind, besteht außerdem keine Notwendigkeit, diese bei einer Implementierung in andere Systeme in diese zu integrieren.

Die Ansteuerung des Mikrocontrollers wird momentan noch mittels ungesichertem HTTP-Request realisiert. Daher kann die Lampe innerhalb des Netzwerks jederzeit durch den Aufruf einer passenden URL gesteuert werden. Da sämtliche Sicherheitsfunktionen zum Schutz der Leuchtmittel auf Mikrocontrollerseite implementiert sind, stellt dies zunächst kein Sicherheitsrisiko dar. Um unautorisierte Bedienung des Simulators zu verhindern, wäre jedoch die Implementierung einer Absicherung des Requests denkbar.

Die Arbeit mit der Arduino IDE war zwar einfach und intuitiv, jedoch fehlen dem integrierten Editor einige Komfortfunktionen und der Flashvorgang braucht relativ lange. Dies liegt am Verhalten der IDE, welche vor jedem Flashvorgang des gesamten Code neu kompiliert. Um dies bei zukünftigen Projekten mit der Arduino-Entwicklungsumgebung zu vermeiden, empfiehlt es sich auf alternative Programmierumgebungen auszuweichen. Hier sei als Beispiel das Projekt ’PlatformIO’ genannt, welches mit einer großen Anzahl an Boards auch außerhalb von Arduino kompatibel ist.

Literaturverzeichnis

[1] “USC Satellite Database,” https://www.ucsusa.org/nuclear-weapons/space-weapons/satellite-database#.WPpdMfkrKHs, Aufgerufen am: 17.6.2018.

[2] Lehrstuhl für Raumfahrttechnik, “RACOON Lab,” https://www.lrt.mw.tum.de/index.php?id=54&L=0, Aufgerufen am: 19.6.2018.

[3] J. Harder, M. Dziura, and S. Haberl, *Future Technologies for Operating Robots in Space*, Paper, Lehrstuhl für Raumfahrttechnik IAC-17,D1,6,7,x38585.

[4] A. Ciadamidaro, “Auslegung und Konstruktion eines Sonnensimulators für eine Simulationsumgebung zur Nahbereichsnavigation von Satelliten,” 2016, Bachelorarbeit, RT-BA 2016/07.

[5] N. Reichenbach, “Entwicklung und Integration eines Erdalbedosimulators zum Testen von Sensoren fur die Nahbereichsnavigation von Satelliten im Erdorbit,” 2017, Semesterarbeit, RT-SA 2017/02.

[6] L. Gössling, “Lichtinstallationen im Architekturbereich - Wie funktioniert die Umsetzung?” 2010.

[7] U. Walter, *Systems Engineering*. Technische Universität München, 2018.

[8] U. Brinkschulte and T. Ungerer, *Mikrocontroller und Mikroprozessoren*. Springer-Verlag, 2010, 3. Auflage, DOI: 10.1007/978-3-642-05398-6.

[9] “What is Arduino?” https://www.arduino.cc/en/Guide/Introduction, Aufgerufen am: 3.10.2018.

[10] B. Evans, *Beginning Arduino Programming*. Apress, 2011, ISBN: 978-1-4302-3777-8.

[11] “Bootloader Development,” https://www.arduino.cc/en/Hacking/Bootloader?from=Tutorial.Bootloader, Aufgerufen am: 31.10.2018.

[12] “Teensy Technical Specifications,” https://www.pjrc.com/teensy/techspecs.html, Aufgerufen am: 4.10.2018.

[13] “Teensy USB Development Board,” https://www.pjrc.com/store/teensy36.html, Aufgerufen am: 6.10.2018.

[14] “NodeMCU v2 - Lua based ESP8266,” https://www.exp-tech.de/module/wireless/wifi/6534/nodemcu-v2-lua-based-esp8266?c=1178, Aufgerufen am: 6.10.2018.

[15] *Datenblatt Mikrocontroller ESP8366EX*, Espressif Systems, Version 5.9.

[16] Zerynth, “Spezifikation und Dokumentation NodeMCU v2,” https://docs.zerynth.com/latest/official/board.zerynth.nodemcu2/docs/index.html, Aufgerufen am: 5.11.2018.

[17] “Temperatursensoren; Grundlagen,” https://www.digikey.de/de/articles/techzone/2011/oct/temperature-sensors-the-basics, Aufgerufen am: 5.10.2018.

[18] “Technische Informationen zu Thermoelementen,” https://www.mts.ch/produkte-messtechnik-schaffhausen-gmbh/guenther-gmbh-temperaturmesstechnik-thermoelemente-widerstandsthermometer/technische-informationen/technische-informationen-zu-thermoelementen/1-aufbau-und-funktionsweise-von-thermoelementen.html/296, Aufgerufen am: 17.10.2018.

[19] “Technische Informationen zu Widerstandsthermometern,” https://www.mts.ch/produkte-messtechnik-schaffhausen-gmbh/guenther-gmbh-temperaturmesstechnik-thermoelemente-widerstandsthermometer/technische-informationen/technische-informationen-zu-widerstandsthermometern/1-aufbau-und-funktionsweise-von-widerstandsthermometern.html/280, Aufgerufen am: 17.10.2018.

[20] Maxim Integrated, *Datenblatt DS18B20 - Programmable Resolution 1-Wire Digital Thermometer*, 2018, 19-7487; Rev 5; 9/18.

[21] “Die Hardware-Uhr,” https://www.ibr.cs.tu-bs.de/users/thuerman/time/rtc.html, Aufgerufen am: 6.10.2018.

[22] “Servo Motors and Control with Arduino Platforms,” https://www.digikey.no/en/articles/techzone/2017/mar/servo-motors-and-control-with-arduino-platforms, Aufgerufen am: 11.10.2018.

[23] “GUI Programming in Python,” https://wiki.python.org/moin/GuiProgramming, abgerufen am: 3.11.18.

[24] “tkinter — Python interface to Tcl/Tk,” https://docs.python.org/3/library/tkinter.html?highlight=tkinter#module-tkinter, Aufgerufen am: 13.10.2018.

[25] W. Doberenz and T. Gewinnus, *Visual C# 2012 – Grundlagen und Profiwissen*. Carl Hanser Verlag GmbH & Co. KG, 2012, DOI: 10.3139/9783446435216.

[26] “Erste Schritte mit JavaScript,” https://developer.mozilla.org/de/docs/Learn/JavaScript/First_steps, Aufgerufen am: 19.10.2018.

[27] “Thermoelement Typ-K (Glasgeflecht, isolierte Edelstahlspitze),” https://www.exp-tech.de/sensoren/temperatur/8174/thermoelement-typ-k-glasgeflecht-isolierte-edelstahlspitze?c=1080, Aufgerufen am: 7.10.2018.

[28] “Adafruit Thermoelement-Verstärker MAX31855 breakout board (MAX6675 upgrade) - v2.0,” https://www.exp-tech.de/sensoren/temperatur/4841/adafruit-thermoelement-verstaerker-max31855-breakout-board-max6675-upgrade-v2.0, Aufgerufen am: 7.10.2018.

[29] “Adafruit DS3231 Precision RTC Breakout,” https://www.exp-tech.de/module/rtc/6917/adafruit-ds3231-precision-rtc-breakout, Aufgerufen am: 7.10.2018.

[30] “Online-Shop Conrad: Modelcraft Standard-Servo RS-22 YMB,” https://www.conrad.de/de/modelcraft-standard-servo-rs-22-ymb-digital-servo-getriebe-material-metall-stecksystem-jr-209154.html, Aufgerufen am: 3.10.18.

[31] Vigor Precision Ltd., *Dateblatt Servo VSD-22YMB*.

[32] HANSMANN Electronic GmbH & Co. KG, *Datenblatt Elektronisches Vorschaltgerät EBM801AC-B(100-240V)*, 2014, Edition 1.0.

[33] USHIO, *Datenblatt Elektronisches Vorschaltgerät EBM3X1*.

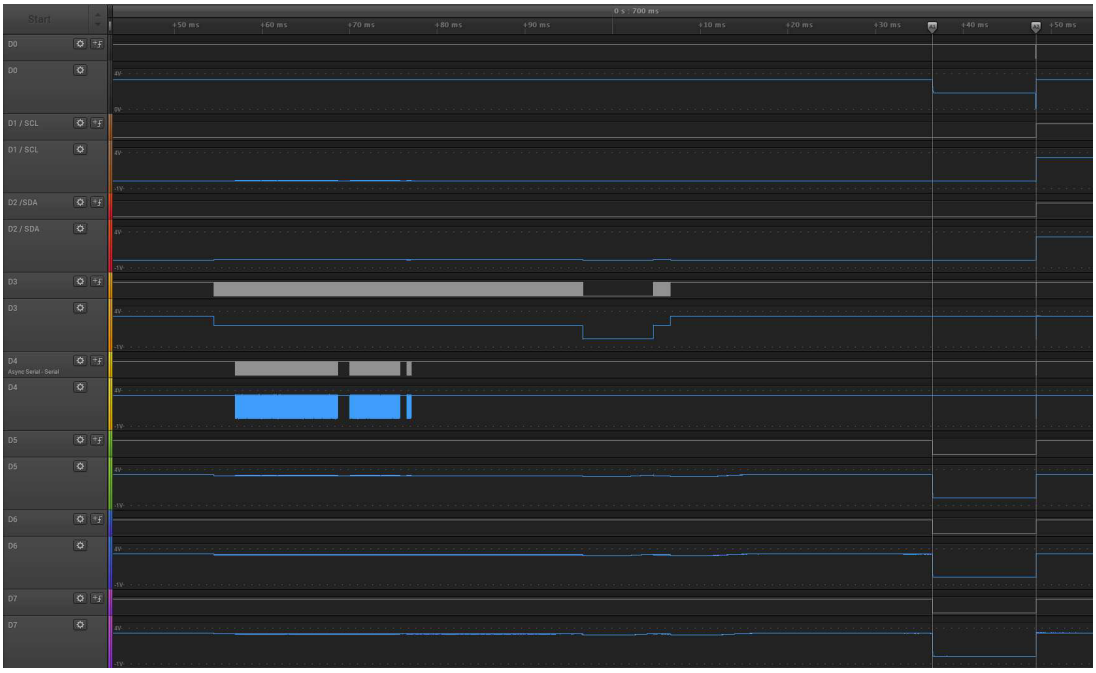
[34] Avago, *Datenblatt Optokoppler ASSR1228*, 2015, aV02-0173EN.

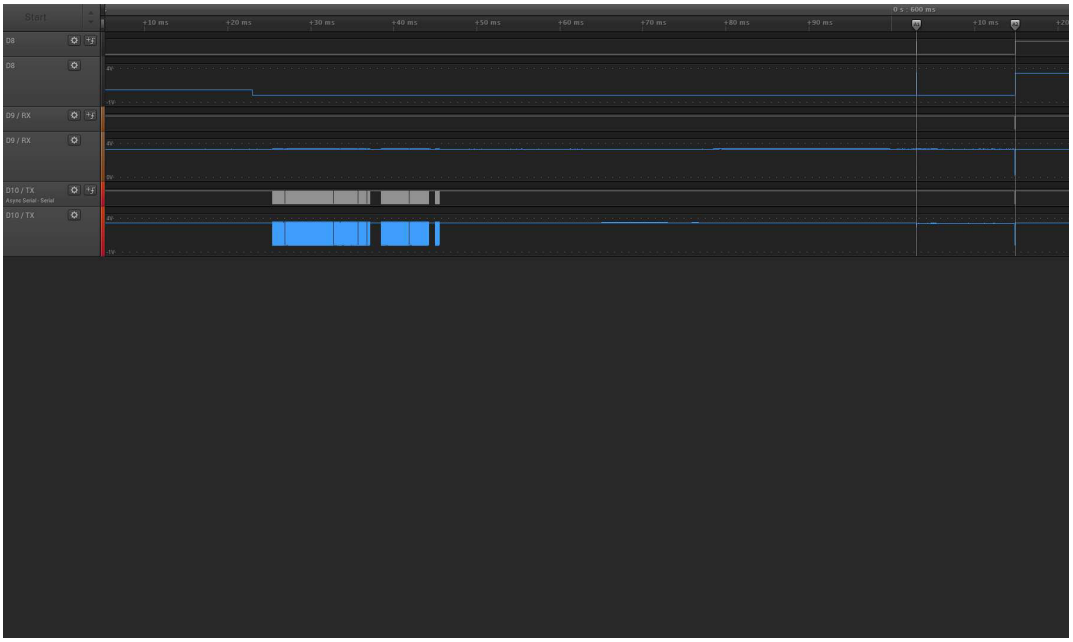
[35] Meanwell, *Datenblatt DR-15-5*, Jun. 2011, dR-15-SPEC.

[36] *Datenblatt LDL1117*, STMicroelectronics, Jul. 2017, DocID030319 Rev 3.

[37] *Datenblatt LD1117*, STMicroelectronics, Nov. 2013, DocID2572 Rev 33.

A Startverhalten des NodeMCU Boards





B Schaltplan