

Betriebliche Praxis

**Bericht über die Erstellung der
Benutzeroberfläche der SIEMENS Comfort
Pannels im SIEMENS TIA-Portal**

Frederik Rath

Matrikelnummer 7202624

30. Juni 2022

Erstprüfer: Prof. Dr.-Ing. Nick Raabe

Zweitprüfer: Dipl.-Ing. Guido Grzeskowiak

Inhaltsverzeichnis

Abkürzungen	III
1 Aufgabenstellung	2
1.1 Analyse der Unterlagen	2
1.2 Grafische Oberfläche	2
2 Analyse und Überarbeitung der Planungsunterlagen	4
2.1 Überarbeitung der Ein- und Ausgangsliste	4
2.2 Ermittlung der eingesetzten Geräte	4
3 Erstellung der Linientopologie	5
4 Umsetzung in TIA-Portal	6
4.1 TIA Portal	6
4.2 Erstellung der Übersicht	6
4.3 Erstellung der Bildbausteine	6
4.4 Animationen der Bausteine	6
4.4.1 Farbbedeutungen	6
4.4.2 Sichtbarkeit von Bausteinen	7
5 Joy-Pi	9
5.1 Aufbau	9
5.2 Raspberry Pi	9
5.3 Sensoren und Aktoren	10
5.3.1 Sensoren	10
5.3.2 Aktoren	11
5.4 Besonderheiten des Joy-Pi	11
5.4.1 Logik	11
5.4.2 Board-Nummern	12
5.4.3 Python-Version	12
6 Realisierung des Projektes	13
6.1 Ansteuerung der Sensoren und Aktoren	13
6.2 Grafische Oberfläche mit Tkinter	15
6.3 TCP/IP-Verbindung	17
7 Ergebnis	18
7.1 Hardware	18
7.2 Grafische Oberfläche	19
7.3 Verbindung	20
8 Zusammenfassung und Ausblick	22
Abbildungsverzeichnis	24
Tabellenverzeichnis	25

Quelltextverzeichnis	26
Literaturverzeichnis	27
A Anhang: Quelltext des Projektes	28
A.1 main_server.py	29
A.2 server.py	30
A.3 main_client.py	33
A.4 client.py	34

Abkürzungen

FU	Frequenzumrichter
GPIO	General-Purpose Input/Output
GUI	Graphical User Interface
IP	Internetprotocol
JSON	JavaScript Object Notation
LCD	Liquid Crystal Display
LED	Light Emitting Diode
PET	Polyethylenterephthalat
SPS	Speicherprogrammierbare Steuerung
TCP/IP	Transmission Control Protocol / Internet Protocol
TIA	Totally Integrated Automation

Einleitung

In dem folgenden Bericht zu der betrieblichen Praxis, welche bei der Firma WPE GmbH in Lünen durchgeführt wurde, geht es um die Erstellung einer Benutzeroberfläche eines SIEMENS Comfort Pannels mit Hilfe des Totally Integrated Automation (TIA)-Portals.

Der Kunde hat für die Ansteuerung einer Mischer- und Abfüllanlage einen neuen Steuerschrank sowie einen Leistungsschrank in Auftrag gegeben. Dabei werden die vorhandenen Bedienelemente demontiert und durch einen im neuen Steuerschrank verbautes Touchpanel ersetzt, zudem wird der vorhandene Leistungsschrank angepasst und erweitert. Für die Motorsteuerung der einzelnen Förderschnecken werden Leitungsschutzschalter, Motorschutzschalter, Leistungsschütze und gegebenenfalls Frequenzumrichter im Leistungsschrank montiert.

Bei der Anlage handelt es sich um ein Greenfieldprojekt, bei dem die komplette Anlage von Grund auf entworfen, gezeichnet, geplant und aufgebaut wird. Den Entwurf der Anlage hat die Firma Günter Anlagenbau übernommen und die erforderlichen technischen Zeichnungen erstellt. Im weiteren Schritt werden die Schaltpläne für die Verdrahtung, die Kabeldimensionierung und die Erstellung der Programmierung von der Firma WPE durchgeführt.

Diese Sortier- und Mischeranlage besteht aus fünf Materialsilos, in die locker geliefertes Material aus LKWs geladen und gelagert wird, zwei BIG-BAG-Stationen, in die aus BIG-BAGs angeliefertes Material in das Verfahren eingeführt werden kann, drei Sortierern, 38 Material transportierende Förderschnecken und fünf Materialmischern. Das Ziel dieser Anlage ist aus geschredderten Einwegflaschen eine sortierte Mischung an Flakes zu erzeugen, die in die Mischer kommt und dort vermengt werden. Im nächsten Prozessschritt werden die vermischten Flakes in den Extruder weitergeleitet und zu neuem Plastikgranulat eingeschmolzen. Um die Farbe des Granulats zu steuern, sortieren die drei Sorter nach unterschiedlichen Prinzipien die Flakes aus. Der Sorter der Marke Tomra besitzt eine Kameratechnik, die in kürzester Zeit die einzelnen Bestandteile erkennen und nach Materialfarbe sortieren und Fremdkörper entfernen kann. Die zwei Unisorter sortieren mit Hilfe eines nahinfrarot Sortiersystems alle hellen Polymerarten und trennen diese voneinander, dabei werden alle dunklen Störstoffe aussortiert. Durch die Förderschnecken können die Sorter gezielt mit Material versorgt werden und so die Farbtöne des gemischten Flakes für jeden der vier Mischern individuell gesteuert werden. Der fünfte Mischer kann als Überlauf verwendet werden, damit die letzten beiden Schnecken nicht voll laufen, falls der Mischer vorher kein Material aus der Förderschnecke bekommt. Der Mischer kann aber auch individuell befüllt werden, sodass es maximal fünf verschiedene Mischungen geben kann.

Diese Art der Wiederaufbereitung von Einwegflaschen kann einen enormen Beitrag dazu beitragen, dass der Verbrauch von Rohöl zur Plastikherstellung sinkt. Gerade in der heutigen Zeit, wo der Umweltschutz und Vermeidung von klimaschädlichen Treibhausgasen verringert und eingeschränkt werden soll. Ein großer Discounter verwendet für die neuen Einwegflaschen aus deren Eigenmarkensortiment nur noch Plastikgranulat aus recycelten Polyethylenterephthalat (PET)-Flaschen. Zudem ist das Granulat sehr begehrt für die Herstellung von anderen Plastikprodukten. Diese Dokumentation befasst sich mit der Planung und der Realisierung des Projektes. Es beginnt mit der Aufgabenstellung. Zu Beginn soll die Analyse der Planungsunterlagen bestehend aus Ein- und Ausgangslisten und Plänen durchgeführt werden. Danach soll eine Topologie der vernetzten Speicherprogrammierbare Steuerung (SPS)-Komponenten erstellt werden, damit die Internetprotocol (IP)-Adressen fest zugewiesen werden können. Abschließend soll eine Visualisierung für das Comfortpannel angefertigt werden.

1 Aufgabenstellung

In diesem Kapitel wird die Aufgabenstellung beschrieben und die Ziele des Projektes definiert.

1.1 Analyse der Unterlagen

Als erstes Ziel gilt es die vorliegenden Unterlagen durchzuarbeiten und zu analysieren. Dabei muss für jedes Betriebsmittel die Anzahl der Ein- und Ausgänge ermittelt werden und diese in die Ein- und Ausgangs Liste dokumentiert werden. Wenn dies abgeschlossen ist müssen mit der Anzahl aller Ein- und Ausgänge die digitalen und analogen Ein- und Ausgangskarten bestellt werden. Als nächstes müssen die Kabelquerschnitte der Zuleitungen für die Motoren berechnet werden. Dies geschieht anhand der Leitungslänge und des Nennstroms im Nennbetrieb, der aus den Leistungen in Abbildung 1.1 berechnet werden kann.

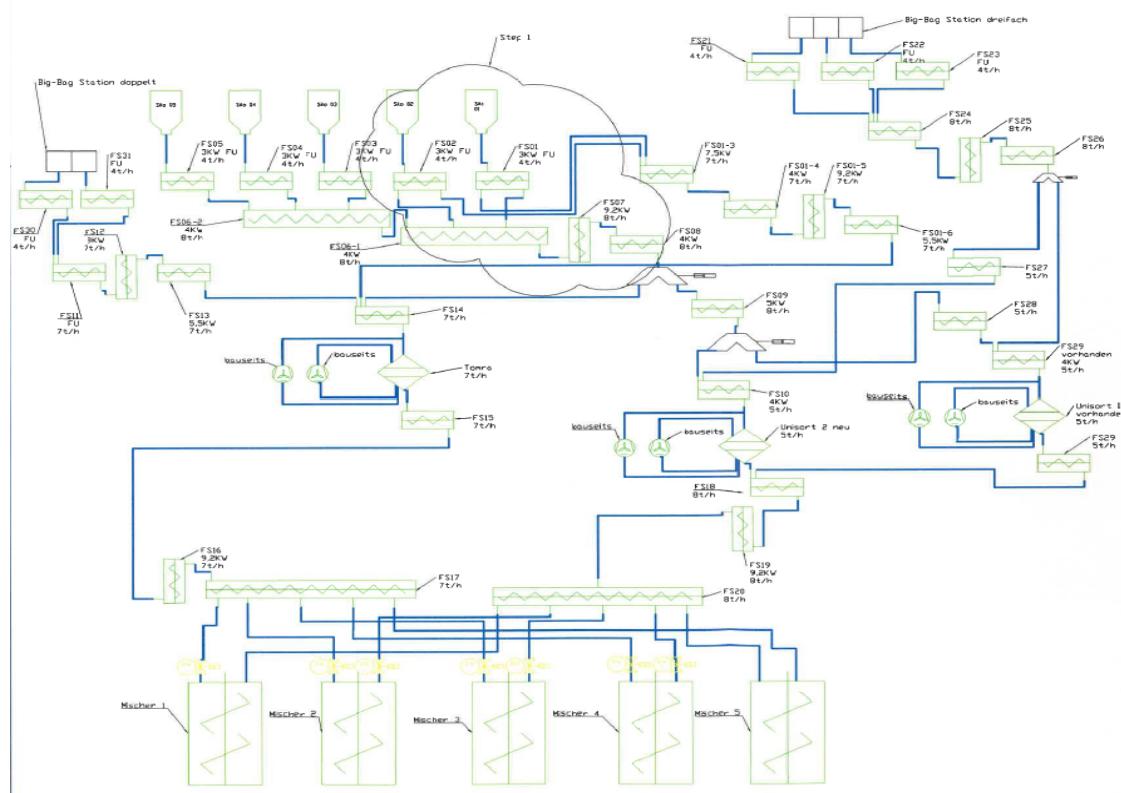


Abbildung 1.1: Übersicht der Antriebe

1.2 Grafische Oberfläche

Nachdem die Analyse der Unterlagen abgeschlossen ist, soll eine grafische Oberfläche im TIA-Portal erzeugt werden. Mit der grafischen Oberfläche sollen alle Motoren gesteuert werden können. Dies bedeutet, dass die Motoren mit Direktantrieb ein- und ausgeschaltet und die Motoren mit einem Frequenzumrichter (FU) in der Drehzahl verändert werden können. Ebenso sollen die Statusmeldungen der Motoren hinzugefügt werden, die zeigen, in welchem Zustand der Motor

befindet. Ab hier wird die Steuerung in Gesamtübersicht, Silos, Big-Bag Stationen, Sorter und Mischer unterschieden. Durch das Tippen auf einzelne Motoren sollen Pop-Up Fenster erscheinen. In diesen Fenstern sollen Informationen zu dem Motor stehen, sich die Bedienung befinden, Störmeldungen angezeigt werden und weitere notwendige Schaltflächen. In der Gesamtübersicht soll keine Bedienung möglich sein. Dies ist nur in den vier anderen Ansichten möglich, zudem soll nur eine eingeschränkte Bedienung möglich sein, wenn der Administrator nicht angemeldet ist. Aus Abbildung 1.1 sollen die Übersichten abgeleitet und erstellt werden.

2 Analyse und Überarbeitung der Planungsunterlagen

In der Analyse der Unterlagen werden die

2.1 Überarbeitung der Ein- und Ausgangsliste

2.2 Ermittlung der eingesetzten Geräte

3 Erstellung der Linientopologie

Damit die Geräte untereinander kommunizieren können, wird eine feste Vergabe der IP-Adressen und eine Linientopologie benötigt. Dabei wird den angeschlossenen Geräten eine feste IP-Adresse vergeben. Die SIEMENS Central prozessing unit (COU) bekommt dabei die erste Adresse zugewiesen, damit von der Position aus die Linie begonnen werden kann. Bei der Linientopologie sind alle Geräte hintereinander verschaltet, da diese Art der Kommunikation untereinander ausreicht und keine komplexen Topologiearten benötigt werden. Die zweite und dritte IP-Adresse bekommt jeweils ein I/O-Modul, da diese am nächsten an der COU sind. Die vierte IP-Adresse bekommt das Comfortpannel, da dies die Bedienstelle ist. Die nächsten IP-Adressen werden nach der Position im Ablauf des Prozesses an die FUs vergeben.

4 Umsetzung in TIA-Portal

4.1 TIA Portal

4.2 Erstellung der Übersicht

4.3 Erstellung der Bildbausteine

4.4 Animationen der Bausteine

4.4.1 Farbbe bedeutungen

Um die verschiedenen Zustände der Betriebsmittel darzustellen, haben die Animationen der Bausteine verschiedene Farben, sodass es für jeden Zustand eine zugeordnete Farbe gibt. In der Tabelle 4.1 sind diese zu sehen.

Bitwert	Farbe	Füllung	Bedeutung
0	grau		Förderschnecke ist inaktiv
1	rot		Förderschnecke ist gestört
2	blau		Förderschnecke ist betriebsbereit
4	gelb		Förderschnecke läuft an
8	grün		Förderschnecke ist im Betrieb
16	gelb		Förderschnecke läuft aus

Tabelle 4.1: Tabelle zur Auflistung der Farben der Motoren

Die Variable, die für die Animation verwendet wird, hat den Datentyp Integer. Dieses Format ergibt immer eine Zahl aus Bits. Die Bitwerte werden so definiert, dass jeweils immer nur ein Bit gesetzt werden muss und gesetzt ist. Dadurch können die Bits von 2^0 bis 2^4 einzeln angesteuert werden. Dies passiert je nach Zustand der Motoren. Ist kein Bit gesetzt färben sich die Motorbausteine grau, dies passiert, wenn die ganze Anlage ausgeschaltet wird oder per NOT-Aus abgeschaltet wird. Ist das 2^0 auf 1 gesetzt, wird der Bildbaustein rot gefärbt. Damit wird die Störung der Förderschnecke angezeigt. Gestört ist die Schnecke, wenn zu viel Material in der Schnecke ist und dadurch die Drehzahl zu niedrig ist und der Drehzahlmelder oder der Motorschutzschalter auslösen. Ebenso ist die Schnecke gestört, wenn der Leitungsschutzleiter ausgelöst hat. Um die Färbung zurückzusetzen muss mit den Quittierungsschaltern der Fehler quittiert werden. Dadurch wird das 2^0 zurückgesetzt und das 2^1 gesetzt. Nun ist Schnecke betriebsbereit und wartet auf das Einschalten durch den Benutzer über das Pannel. Wird die Schnecke eingeschaltet wird das 2^1 wieder zurückgesetzt. Nun kommt es darauf an, ob der Motor mit einem FU oder per Direktantrieb eingeschaltet wird. Sollte er per FU eingeschaltet werden, wird das 2^2 gesetzt und der Motor färbt sich gelb. Dies bedeutet, dass der Motor mit dem Hochlaufen beginnt. Schafft er dies nicht in einer vorgegebenen Zeit tritt eine Störung auf und die rote Färbung erscheint erneut. Ist das Hochlaufen absolviert wird das 2^2 Bit zurückgesetzt und das 2^3 Bit gesetzt und der Motor wird grün gefärbt. Dies passiert ohne FU direkt, da es kein Hochlauf des Motors gibt. Sobald der Motor ausgeschaltet wird wird das Bit zurück gesetzt und der Motor ist erneut blau, da dann in die Betriebsbereitschaft versetzt wird. Sollte ein FU vorhanden sein wird für das runterlaufen des

Motors das 2^4 Bit gesetzt und der Motor leuchtet erneut gelb bis er aus ist und wie der Motor ohne FU wieder in Betriebsbereitschaft ist.

Bitwert	Farbe	Füllung	Bedeutung
0	grau		Rohr ist inaktiv (Anlage ist ausgeschaltet)
1	rot		Förderschnecke ist gestört, kein Transport
2	hellgrün		Rohr wartet auf Flakes
4	dunkelgrün		Flakes werden transportiert

Tabelle 4.2: Tabelle zur Auflistung der Farben der Motoren

In Tabelle 4.2 sind die Färbungen der Rohre aufgezeigt. Ein Rohr hat dabei vier mögliche Zustände, die von den Bits 2^0 bis 2^2 definiert werden können. Ist kein Bit gesetzt ist das Rohr grau eingefärbt. Dies passiert, wenn die Anlage ausgeschaltet wurde und keine Komponenten mehr aktiv sind. Wird das 2^0 Bit gesetzt ist eine Störung aufgetreten, da die Rohre mit den Motoren verknüpft sind, gibt es immer eine Störung, wenn die vorherige Förderschnecke gestört ist. Befindet sich die Förderschnecke in Betriebsbereitschaft wird das 2^1 Bit gesetzt und das Rohr ist hellblau gefärbt. Fördert die Schnecke Material durch das Rohr oder läuft hoch oder runter, wird das 2^2 Bit gesetzt. Sobald Material von der Förderschnecke bewegt wird, wird das Rohr dunkelgrün gefärbt, damit der Materialfluss deutlich nachvollzogen werden kann.

4.4.2 Sichtbarkeit von Bausteinen

Neben der Färbung gibt es auch die Möglichkeit die Sichtbarkeit von Bausteinen und Linien von einer Variablen abhängig zu machen. Dies ist besonders bei wenig Platz von Vorteil, da nicht benötigte Elemente einfach ausgeblendet werden können. In der Abbildung 4.1 ist es deutlich zu sehen wie eng die Linien aneinander vorbei laufen.

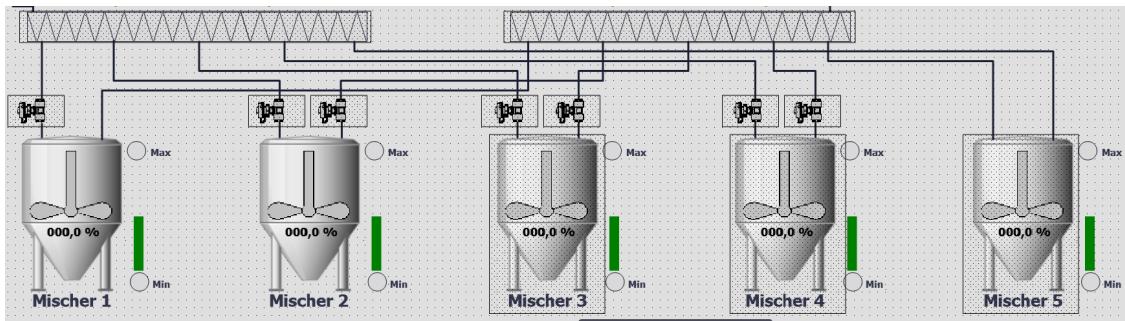


Abbildung 4.1: Zu enge Rohrdarstellung

Damit nun keine Verwirrung durch zu viele verschiedenen farbige Linien auf einen Fleck entsteht, werden die nicht aktiven Linien ausgeblendet. Dafür kann in den Eigenschaften die Sichtbarkeit eingestellt werden. Denn nur das aktive Rohr soll sichtbar sein. In Abbildung 4.2 ist zu sehen, welche Variable dafür benötigt wird.

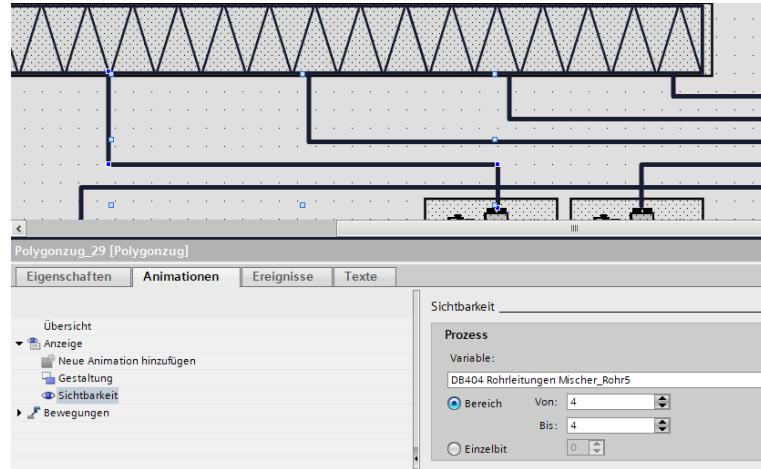


Abbildung 4.2: Verwendung der Variable für die Sichtbarkeit

Die „Variable DB404 Rohrleitung Mischer_Rohr5“ ist ein Integer der aus dem Datenbaustein 404 abgelesen wird. In dem Bereich des Bitwertes von vier bis vier ist diese Rohrleitung sichtbar. Der Wert tritt auf sobald die Förderschnecke und das Ventil zum Mischer aktiv sind. Ist dies nicht der Fall bleibt die Linie unsichtbar. In Abbildung 4.2 ist sind die engen Rohrverläufe ausgeblendet und nur noch eine Linie sichtbar, damit keine Verwirrung entsteht.

Die Sichtbarkeit spielt auch bei den Pop-Ups eine große Rolle. In diese sind so viele Informationen und Funktionen eingebettet, dass durch Überlappungen nichts erkannt oder bedient werden kann. In diesem Fall verteilt man die zusammengehörigen Bausteine auf verschiedene Ebenen. Durch das Betätigen einer Schaltfläche aus dem Menü wird ein Merker gesetzt. Über diese Merker kann dann eingestellt werden, welche Bausteine Sichtbar sind und nur wenn die diese sichtbar sind können sie auch bedient werden.

5 Joy-Pi

Dieses Kapitel behandelt den Joy-Pi-Koffer. Es soll gezeigt werden woraus der Koffer besteht und welche Besonderheiten er hat.

5.1 Aufbau

Der Joy-Pi-Koffer ist ein Entwickler-Board, welches aus diversen Sensoren, Aktoren und einem eingebautem Touch-Display besteht. Außerdem befindet sich in dem Koffer ein Raspberry Pi. im Folgenden wird in Abbildung 5.1 der im Projekt verwendete Koffer gezeigt.

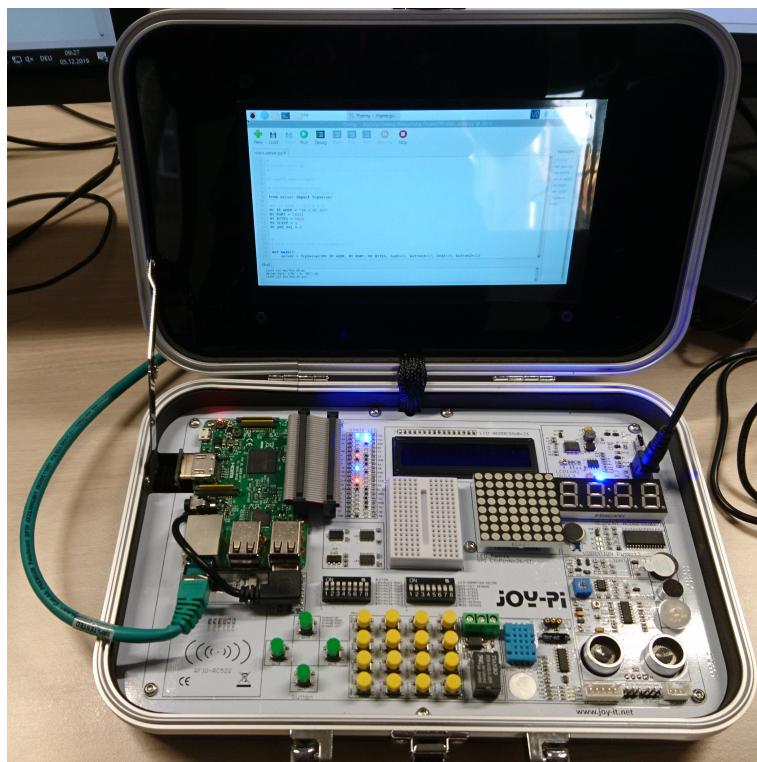


Abbildung 5.1: Der Joy-Pi-Koffer

5.2 Raspberry Pi

Der Raspberry Pi ist ein kleiner Computer, der auf dem Joy-Pi montiert ist (siehe Abbildung 5.2). Es gibt verschiedene Versionen des Raspberry Pi. In diesem Projekt wird der Raspberry Pi 3 B verwendet auf dem das Linux-Betriebssystem Raspbian installiert ist. Eine Programmierumgebung für Python mit dem Namen Thonny ist vorinstalliert. Es ist eine sehr einfach gehaltene Umgebung ohne viele Extras. Sie dient zur schnellen Programmierung, bietet allerdings wenig Unterstützung bei Problemen. Die technischen Daten des Raspberry Pi, wie Prozessorleistung etc. sind hier [9] nachzulesen.



Abbildung 5.2: Der Raspberry Pi 3 B

5.3 Sensoren und Aktoren

Wie schon erwähnt besteht der Koffer aus diversen Sensoren und Aktoren. In zwei Unterkapiteln wird erläutert welche Sensoren und Aktoren verbaut sind und kurz erklärt, wie sie funktionieren. Mit folgender Grafik 5.3 wird gezeigt welche Bauteile sich auf dem Koffer befinden. Die Bauteile sind blau für die Sensoren und rot für die Aktoren. Es gibt auf dem Bild zwei Nummerierungen, die in unterschiedlichen Farben aufgeführt sind, um Verwechslungen zwischen Sensoren und Aktoren zu vermeiden.

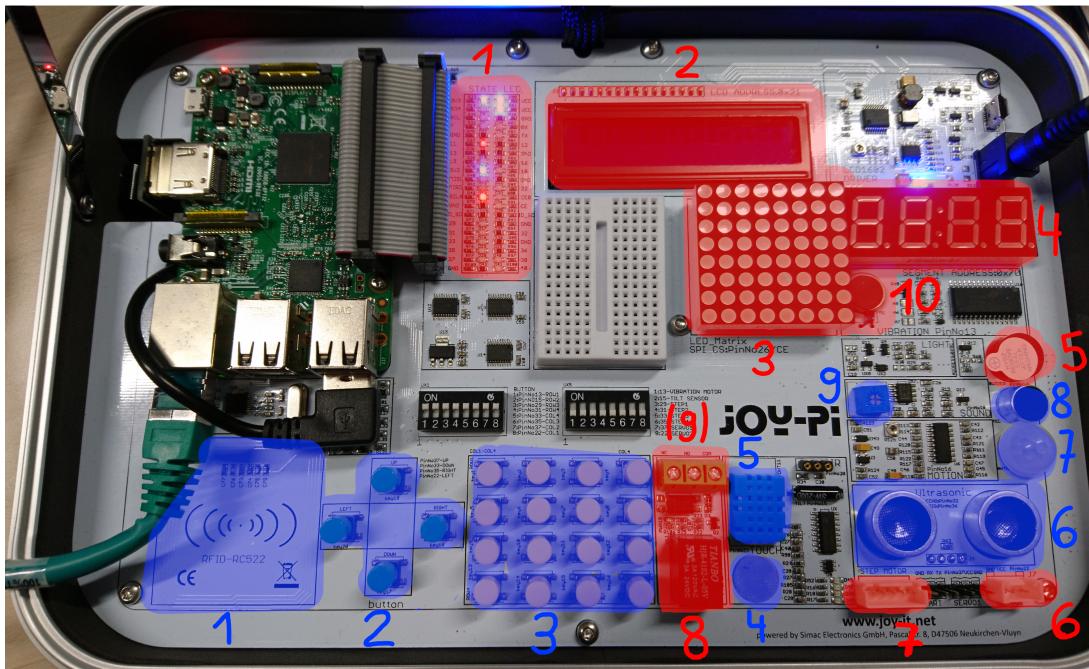


Abbildung 5.3: Sensoren und Aktoren des Joy-Pi

5.3.1 Sensoren

In diesem Unterkapitel werden die Sensoren behandelt. Mithilfe der folgenden Tabelle werden Name und Funktion der Sensoren erläutert. Die Nummerierung stammt aus Abbildung 5.3.

Nummer	Sensor	Funktion
1	RFID-Modul	Zum Scannen eines RFID-Tags

Nummer	Sensor	Funktion
2	Buttons mit Light Emitting Diode (LED)	Signal durch Tasten erzeugen
3	Buttons ohne LED	Signal durch Tasten erzeugen
4	Berührungssensor	Signal durch Berührung erzeugen
5	Luftfeuchtigkeits-/Temperatursensor	Kann die Luftfeuchtigkeit und Temperatur der Umgebung messen
6	Ultraschallsensor	Wird zur Abstandsmessung verwendet
7	Bewegungssensor	Erkennt Bewegung in der Umgebung
8	Schallsensor	Erkennt Geräusche in der Umgebung
9	Potentiometer	Ein per Hand einstellbarer Widerstand

Tabelle 5.1: Tabelle zur Auflistung der Sensoren

5.3.2 Aktoren

In diesem Unterkapitel werden die Aktoren behandelt. Mithilfe der folgenden Tabelle werden Name und Funktion dieser erläutert. Die Nummerierung stammt aus Abbildung 5.3.

Nummer	Aktor	Funktion
1	LEDs	Anzeigen von Signalen durch Leuchtzustand
2	Liquid Crystal Display (LCD)-Anzeige	Anzeigen von Daten (gute Auflösung)
3	LED-Matrix	Anzeigen von Daten (schlechte Auflösung)
4	Siebensegment-Anzeige	Analoge Werte können angezeigt werden
5	Buzzer	Erzeugt Töne bei Aktivierung
6	Anschluss Servomotor	Schrittweise einstellbarer Motor
7	Anschluss Schrittmotor	Motor mit hoher Drehzahl
8/(9)	Relais mit Anschlüssen	Öffnen und Schließen von elektrischen Schaltkreisen
10	Vibrationsmodul	Vibriert bei Aktivierung

Tabelle 5.2: Tabelle zur Auflistung der Aktoren

Zusätzlich können die Informationen zu Sensoren und Aktoren in der Anleitung des Joy-Pi nachgelesen werden [1].

5.4 Besonderheiten des Joy-Pi

Der Joy-Pi hat diverse Besonderheiten, die in diesem Teil des Berichts erläutert werden. Es wird auf Themen wie Logik, Board-Nummern und die Python-Version eingegangen.

5.4.1 Logik

In diesem Projekt werden neben einer grafischen Oberfläche und einer Transmission Control Protocol / Internet Protocol (TCP/IP)-Verbindung auch Hardware-Komponenten programmiert. Diese Komponenten erzeugen und verarbeiten Signale, die korrekt interpretiert und verarbeitet werden müssen. Mithilfe von Tastern (Hardware-Eingänge) sollen LEDs (Hardware-Ausgänge)

an- und ausgeschaltet werden. Daraus folgt, dass die Ein- und Ausgänge digital sind. Nun ist bei dem Joy-Pi darauf zu achten, dass die digitalen Ein- und Ausgänge mit einer negativen Logik arbeiten. Somit wird die Logik vom Board vorgegeben und nicht vom Raspberry Pi. Das bedeutet, dass wenn zum Beispiel an einem Ausgang eine positive Spannung anliegt (*High-Signal*), wird die LED nicht ein- sondern ausgeschaltet. Um sie einzuschalten, müsste die LED den Wert *Low* erhalten (niedriger Spannungspegel). Bei den Eingängen funktioniert dies auf die gleiche Art, allerdings werden hierbei nur die Flanken von ihnen abgefragt (näheres wird in Kapitel 3 beschrieben).

5.4.2 Board-Nummern

Die Bauteile des Joy-Pi sind an Pins angeschlossen, welche auf dem Board mit Nummern versehen sind. Diese Nummerierung unterscheidet sich von der in der Anleitung. In diesem Projekt wird nicht mit den Nummern in der Anleitung gearbeitet, sondern mit den Board-Nummern. Wenn also im Quelltext eine Pin-Nummer zugewiesen wird, muss auf dem Board nachgeschaut werden, um welchen Pin es sich handelt. Die Abbildung 5.4 zeigt wie die Nummerierung dargestellt ist. Es sind Nummern, die neben den jeweiligen Bauteilen angebracht sind. Hier werden beispielsweise die LEDs gezeigt.

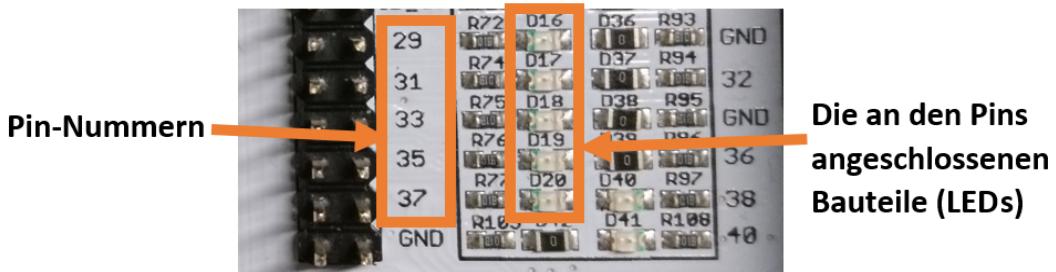


Abbildung 5.4: Pin-Nummern und deren Bauteile

Aus der Abbildung geht nicht hervor, dass es Pins gibt, die an mehreren Bauteilen angeschlossen sind. Zum Beispiel Pin 37 aus Abbildung 5.4, welcher zum einen an die LED angeschlossen ist, aber auch an einen Taster.

5.4.3 Python-Version

Die Programmiersprache Python ist eine interpretierbare Sprache und hat verschiedene Versionen. Je nach Version unterscheidet sich die Syntax (Grammatik der Programmiersprache), außerdem bieten aktuellere Versionen neue Funktionen mit denen gearbeitet werden kann. In diesem Projekt wird Python 3.7.3 zur Programmierung genutzt.

6 Realisierung des Projektes

Folgend wird die Umsetzung der zuvor geplanten Aspekte erläutert.

6.1 Ansteuerung der Sensoren und Aktoren

Wie zuvor erwähnt wurde, soll eine LED durch einen Hardware-Taste ein- und ausgeschaltet werden können. Dies gelingt durch die Ansteuerung der Hardware-Pins des Raspberry Pis, welche General-Purpose Input/Output (GPIO) genannt werden. An diesen Pins sind Aktoren und Sensoren angeschlossen. Die Ansteuerung erfolgt in Python über die GPIO-Bibliothek [6]. Sensoren sind im Rahmen dieses Projektes die Hardware-Tasten des Joy-Pi, welche in Abbildung 6.1 dargestellt sind. In Abbildung 6.2 werden die verwendeten Aktoren, hier LEDs, gezeigt.

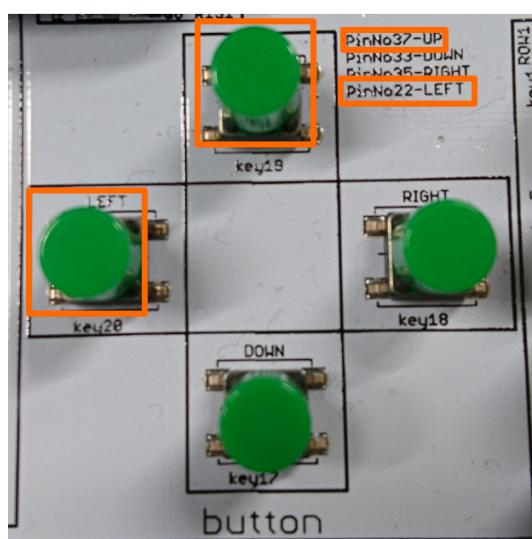


Abbildung 6.1: Im Projekt verwendete Tasten des Joy-Pi

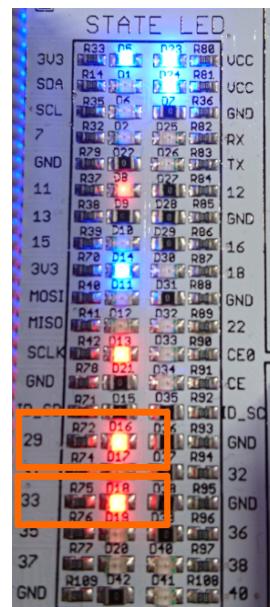


Abbildung 6.2: Im Projekt verwendete LEDs des Joy-Pi

Es werden die markierten Tasten und LEDs verwendet. Die obere Taste ist an Pin 37 angeschlossen (vgl. Abbildung 6.1) und soll die Beleuchtung im Raum 1 steuern. Sie wird im Folgenden Text *Taste 1* genannt. Die linke Taste soll das Licht in Raum 2 über Pin 22 (vgl. Abbildung 6.1) ansteuern und wird daher im weiterem Verlauf *Taste 2* benannt. Die obere Leuchtdiode ist an Pin 29 angeschlossen und simuliert das Licht des ersten Raumes. Die untere LED stellt die Beleuchtung des zweiten Raumes dar und erhält über Pin 33 ihr Signal (siehe Abbildung 6.2). Der Aktor des ersten Raumes wird im Textverlauf *LED 1*, der des zweiten Raumes *LED 2* genannt. Über die GPIO Klasse können Zustände dieser Pins eingelesen und Signaländerungen ausgegeben werden.

Der Quelltext 6.1 zeigt, wie in Python ein Tastendruck eingelesen wird. Nachdem in der 2. Zeile des Quelltextes 6.1 einer Variablen `button1_pin` die Pin-Nummer 37 (*Taste 1*) der Hardware-Pins zugeordnet wurde, wird in Zeile 5 diese Pin-Nummer der Methode `setup` der GPIO Klasse

übergeben und über `GPIO.IN` als Eingang festgelegt. Der Parameter `GPIO.PUD_DOWN` definiert an dem übergebenen Pin einen Pulldown-Widerstand. Bei geöffnetem Kontakt wird also die Spannung am Pin auf Masse gezogen und erzeugt so ein eindeutiges LOW-Signal [10]. In Zeile 8 des Quelltextes 6.1 kann über die Methode `add_event_detect` definiert werden, welche Aktion bei Betätigung der Taste durchgeführt werden soll. Dabei wird die Pin-Nummer, die Flanke bei der die Aktion durchgeführt werden soll (hier die steigende Flanke), die Callback-Funktion und eine Zeit übergeben. Die Callback-Funktion ist dabei die Funktion, die bei Betätigung ausgeführt wird (hier: `button1_interrupt`). Die Zeit von 500 ms entprellt die Taste und verhindert so unerwünschte Signale. Die Funktion `button1_interrupt` soll *LED 1* steuern. Der Quelltext lässt sich dann auf *Taste 2* übertragen.

```
1 #Erstellen einer Variablen
2 button1_pin=37
3
4 #Festlegung als Eingang
5 GPIO.setup(button1_pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
6
7 #Ereignis bei Betätigung des Buttons
8 GPIO.add_event_detect(button1_pin, GPIO.RISING, callback=button1_interrupt,
bouncetime=500)
```

Quelltext 6.1: Python-Code zum Einlesen des Tastenzustandes

Wie eine LED in Python angesteuert wird, zeigt Quelltext 6.2. Pin 33 (*LED 1*) wird in Zeile 2 des Quelltextes 6.2 die Variable `LED1_pin` zugewiesen. Folgend wird in Zeile 5 diese Variable über `GPIO.OUT` als Ausgang festgelegt. Aufgrund der internen Verschaltung wird die LED in negativer Logik angesprochen (Näheres ist in Kapitel 5.4.1 erklärt). Aus diesem Grund wird in Zeile 8 über `GPIO.LOW` eine Spannung an den `LED_pin` angelegt: Die LED leuchtet. Über `GPIO.HIGH` wird dies in Zeile 9 wieder rückgängig gemacht (siehe auch Kapitel 5.4.1). Auch hier lässt sich der Quelltext auf die zweite Leuchtdiode (*LED 2*) übertragen (vgl. Abbildung 6.2).

```

1 #Erstellen einer Variablen
2 LED1_pin=33
3
4 #Festlegung als Ausgang
5 GPIO.setup(LED1_pin, GPIO.OUT)
6
7 #Ansteuerung
8 GPIO.output(LED_pin, GPIO.LOW)
9 GPIO.output(LED_pin, GPIO.HIGH)

```

Quelltext 6.2: Python-Code zur Ansteuerung einer LED

Die oben genannten Errungenschaften können nun programmiertechnisch so verknüpft werden, dass mit den Tasten die LEDs angesteuert werden. Nun ist es möglich mit *Taste 1 LED 1* und mit *Taste 2 LED 2* anzusteuern.

6.2 Grafische Oberfläche mit Tkinter

Die Ansteuerung der LEDs soll nun auch über eine grafische Oberfläche möglich sein. Diese soll zunächst auf dem Raspberry Pi aufgerufen werden können. Um das Graphical User Interface (GUI) zu erstellen, wird die Python-Bibliothek Tkinter [2] verwendet. In Abbildung 6.3 ist diese grafische Oberfläche dargestellt. Jeder Raum besitzt jeweils eine Taste zum ein- und ausschalten sowie eine LED-Status-Anzeige. Diese soll darstellen, ob die Leuchtdiode auf dem Joy-Pi eingeschaltet ist oder nicht.

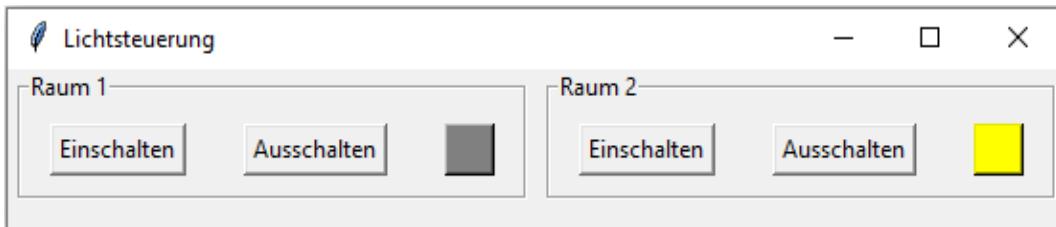


Abbildung 6.3: Grafische Oberfläche programmiert mit Python und Tkinter

In Quelltext 6.3 ist die programmiertechnische Beschreibung des GUI dargestellt. Dabei wird in Zeilen 3 bis 5 ein Fenster mit der Überschrift `Lichtsteuerung` und den Maßen `550 px · 80 px` erstellt. Der erste Rahmen, der die Schaltflächen für Raum 1 beinhaltet (vgl. Abbildung 6.3) wird in den Zeilen 7 und 8 erstellt. Die Schaltfläche zum Einschalten wird in Zeilen 10 und 11, die zum Ausschalten in Zeilen 13 und 14 generiert. Zusätzlich entsteht durch die 16. und 17. Zeile eine Schaltfläche, die neben der Taste zum Ausschalten positioniert wird, um den Zustand der LEDs anzuzeigen. Diese Schritte werden für Raum 2 in Zeilen 19-30 wiederholt.

```

1 import tkinter as tk
2

```

```
3 window = tk.Tk()
4 window.title("Lichtsteuerung")
5 window.geometry("550x80")
6
7 lbl= tk.LabelFrame(window, text="Raum 1", width=250, height=80)
8 lbl.grid(row=0, column=0, columnspan=2, sticky="W", padx=5, pady=0, ipadx=0,
9           ipady=0)
10 an_btn = tk.Button(lbl, text="Einschalten", command=callback_an_raum1)
11 an_btn.grid(column=0, row=1, padx=15, pady=10)
12
13 aus_btn = tk.Button(lbl, text="Ausschalten", command=callback_aus_raum1)
14 aus_btn.grid(column=1, row=1, padx=15, pady=10)
15
16 anzeigen1 = tk.Button(lbl, text="      ", bg="grey", command="")
17 anzeigen1.grid(column=2, row=1, padx=15, pady=10)
18
19 lbl2= tk.LabelFrame(window, text="Raum 2", width=250, height=80)
20 lbl2.grid(row=0, column=2, columnspan=2, sticky="W", padx=5, pady=0, ipadx=0,
21           ipady=0)
22 an_btn = tk.Button(lbl2, text="Einschalten", command=callback_an_raum2)
23 an_btn.grid(column=0, row=1, padx=15, pady=10)
24
25 aus_btn = tk.Button(lbl2, text="Ausschalten", command=callback_aus_raum2)
26 aus_btn.grid(column=1, row=1, padx=15, pady=10)
27
28 anzeigen2 = tk.Button(lbl2, text="      ", bg="grey", command="")
29 anzeigen2.grid(column=2, row=1, padx=15, pady=10)
```

Quelltext 6.3: Python-Code der verwendeten Dictionaries

Das Schalten der Tasten eines Raumes bewirkt die Farbänderung der LED-Status-Anzeige. Diese wird beim Einschalten gelb und beim Ausschalten grau (vgl. Abbildung 6.3). Durch die Callback-Funktion, welche durch den Parameter `command` übergeben wird, kann dies erreicht werden (siehe Quelltext 6.2 Zeilen 10, 13, 22 und 25). Um nun auch die Leuchtdioden auf dem Joy-Pi mit den Schaltflächen der GUI ansteuern zu können, wird in die jeweilige Callback-Funktion der Befehl zum Schalten der LEDs eingegeben.

Nun können durch die grafische Oberfläche die Leuchtdioden ein- bzw. ausgeschaltet und der Zustand der LEDs angezeigt werden.

6.3 TCP/IP-Verbindung

Die Ansteuerung der Leuchtdioden kann nun über die grafische Oberfläche auf dem Raspberry Pi erfolgen. Das nächste Ziel ist, das GUI auf einem anderen Computer zu starten, um von dort die Befehle an den Raspberry Pi zu übertragen. Damit dies gelingt, soll die Verbindung über TCP/IP hergestellt werden. Die TCP/IP-Verbindung gehört zu der Familie der Netzwerk-Protokolle und ermöglicht einen Austausch von Datenpaketen über eine Netzwerkverbindung [8]. In Abbildung 6.4 wird diese Verbindung schematisch dargestellt. Dabei soll der Joy-Pi und damit der Raspberry Pi die Aufgabe des Servers übernehmen. Der externe Computer stellt den Client dar und soll sich mit dem Server verbinden. Dafür wird dem Client eine Adresse vorgegeben, die auf dem Prinzip des IP beruht. Es ist die IP-Adresse des Raspberry Pi. Dieser wartet darauf, dass sich ein Client mit ihm verbindet. Anschließend können vom Raspberry Pi Informationen über die Tasten und Zustände der LEDs an den externen Computer gesendet werden. Dieser wiederum sendet die Signale der Schaltflächen an den Server. Dies geschieht gleichzeitig.



Abbildung 6.4: Datenaustausch zwischen externem Computer und Joy-Pi über TCP/IP-Verbindung

Die Verbindung wird in Python unter Verwendung des Moduls `json` [4] umgesetzt. JavaScript Object Notation (JSON) ist eine Syntax, die zum Datenaustausch oder zur Datenspeicherung dient [3]. Dabei ähnelt diese Syntax sehr dem der Python-Dictionaries. Durch Funktionen aus dem `json`-Modul können diese Dictionaries ausgelesen oder erzeugt werden [5]. In diesem Projekt werden Dictionaries verwendet, um die Zustände der LEDs, der Tasten und der Schaltflächen des GUI zu übertragen. Dieser Datenaustausch wird durch die Python-Bibliothek `socket` [11] ermöglicht. Die Dictionaries werden durch die Methode `json.dumps` in Strings umgewandelt [5] und können über `encode()` (eine Funktion der Klasse `socket`) als Byte-Kette abgesendet werden [7]. Beim Empfänger wird diese Byte-Kette über `decode()` zurück in einen String umgewandelt, um daraus anschließend mit `json.loads` ein Dictionary zu erzeugen [5]. Folgend kann ein Auslesen und Auswerten der Daten erfolgen.

7 Ergebnis

Das Ergebnis des Projektes wird im Folgenden erläutert.

7.1 Hardware

Im Projekt wurde ein Computer verwendet (vgl. Abbildung 7.1 links), der über das Netzwerk mit dem Raspberry Pi (vgl. Abbildung 7.1 rechts) verbunden ist. An diesem sind sowohl die verwendeten LEDs (rot markiert) als auch die genutzten Tasten (blau markiert) angeschlossen.

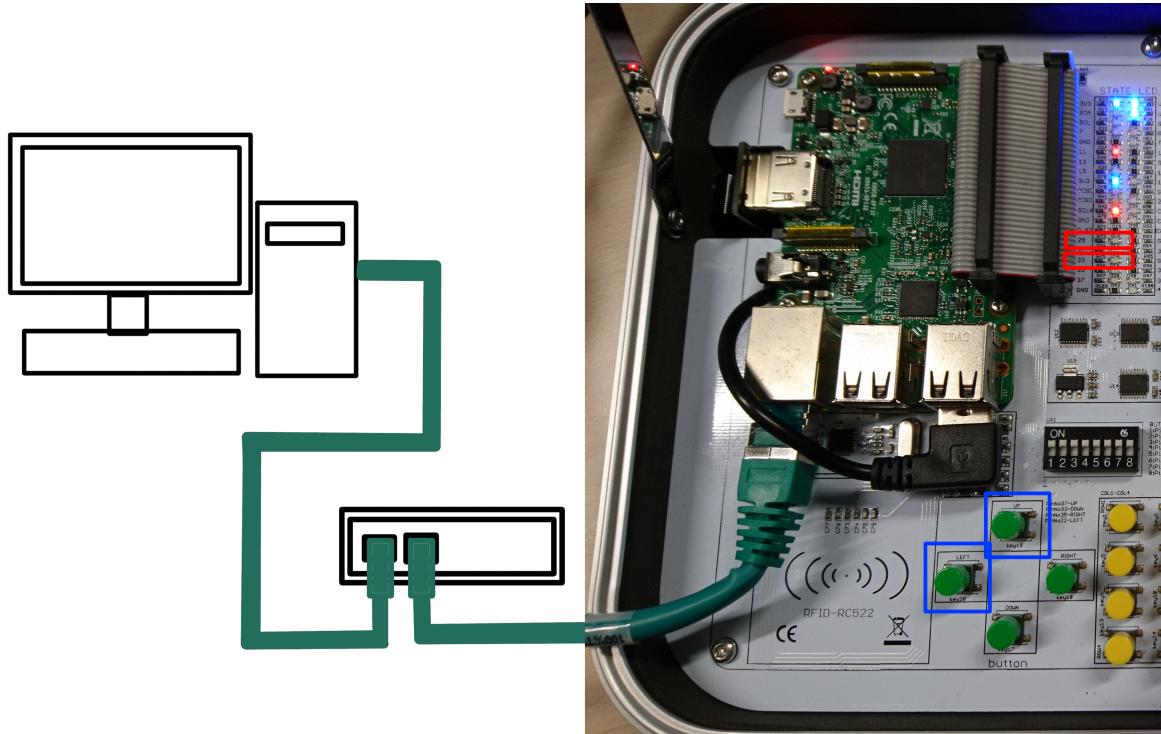


Abbildung 7.1: Projektaufbau mit genutzter Hardware

In Abbildung 7.2 werden die unterschiedlichen Zustände gezeigt, die *LED 1* und *LED 2* annehmen können. Wird keine der beiden Tasten gedrückt, leuchtet auch keine der LEDs (vgl. linkes Bild aus Abbildung 7.2). Wird *Taste 1* betätigt, wird *LED 1* eingeschaltet (vgl. mittleres linkes Bild aus Abbildung 7.2). Sobald anschließend *Taste 2* gedrückt wird, leuchtet zusätzlich noch *LED 2* (vgl. mittleres rechtes Bild aus Abbildung 7.2). Wird *Taste 1* erneut gedrückt erlischt *LED 1* und nur noch *LED 2* bleibt eingeschaltet (vgl. rechtes Bild aus Abbildung 7.2). Diese erlischt, sobald *Taste 2* erneut gedrückt wird. Diese Schaltvorgänge können beliebig oft und in beliebiger Reihenfolge durchgeführt werden.

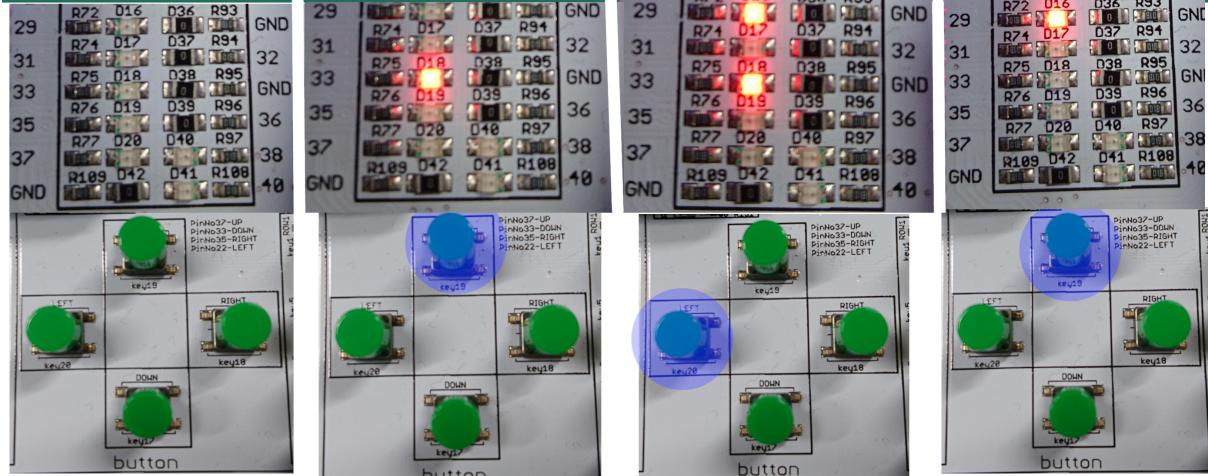


Abbildung 7.2: Zustände der LEDs

7.2 Grafische Oberfläche

Um die LEDs auch von einem anderen Computer als den Raspberry Pi aus steuern zu können, wurde eine grafische Oberfläche programmiert (vgl. Kapitel 6.2). Diese kann nun über einen externen Rechner aufgerufen werden. In den folgenden Abbildungen ist das GUI bei unterschiedlichen Betätigungen dargestellt.

In Abbildung 7.3 wird über die grafische Oberfläche *Raum 1* eingeschaltet. Dabei wird über die LED-Status-Anzeige mitgeteilt, dass *LED 1* auf dem Joy-Pi leuchtet. Wird anschließend in Abbildung 7.4 wieder ausgeschaltet, erlischt *LED 1* und die Schaltfläche der Status-Anzeige wird grau.

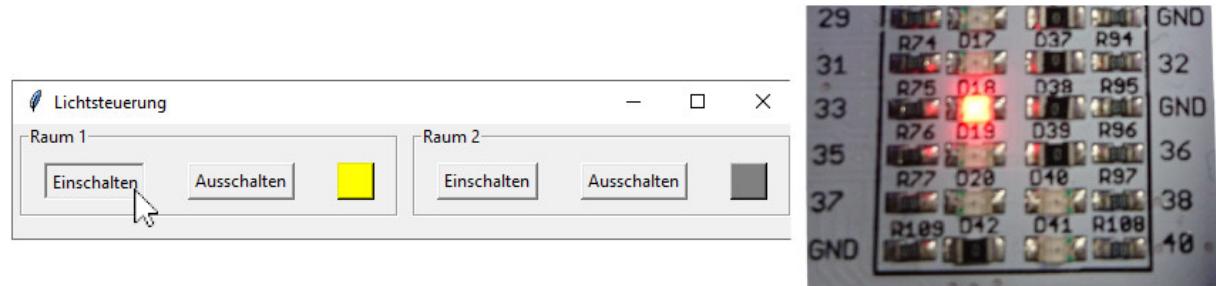


Abbildung 7.3: Grafische Oberfläche bei Betätigung der Einschalt-Taste des ersten Raumes

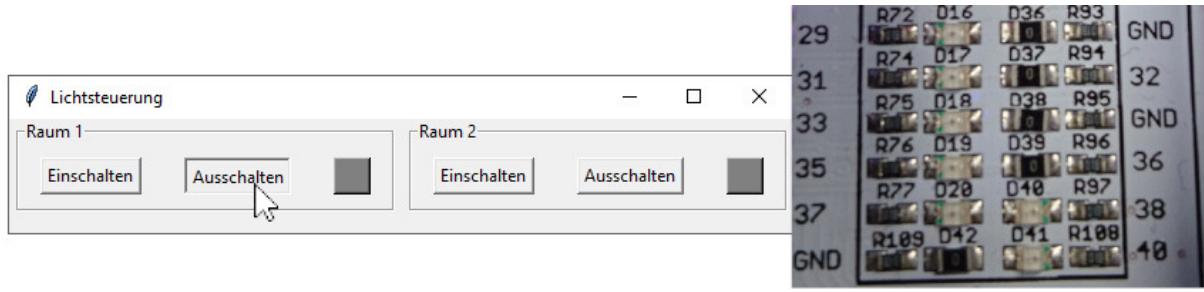


Abbildung 7.4: Grafische Oberfläche bei Betätigung der Ausschalt-Taste des ersten Raumes

Genauso verhält es sich auch beim Ein- und Ausschalten der *LED 2* des *Raumes 2*. Beim Einschalten leuchtet *LED 2* und die Status-Anzeige wird gelb (vgl. Abbildung 7.5), beim Ausschalten erlischt *LED 2* und der LED-Status wechselt auf grau (vgl. Abbildung 7.6).

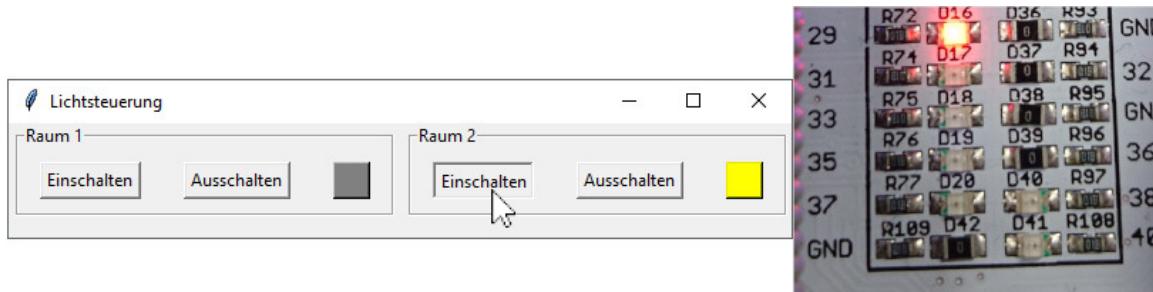


Abbildung 7.5: Grafische Oberfläche bei Betätigung der Einschalt-Taste des zweiten Raumes

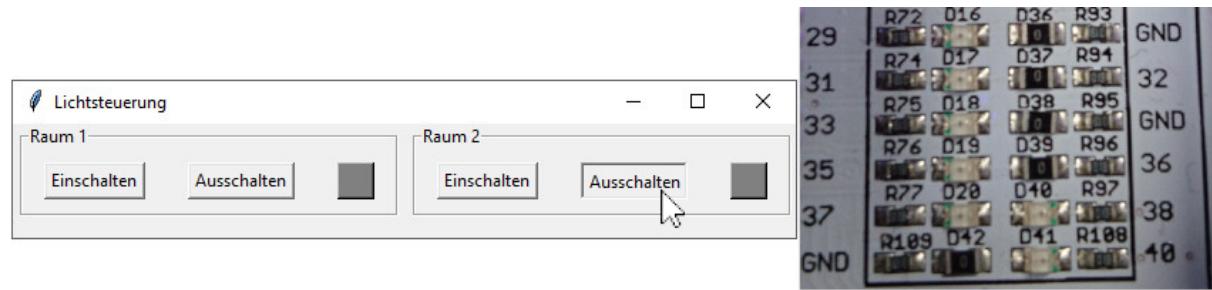


Abbildung 7.6: Grafische Oberfläche bei Betätigung der Ausschalt-Taste des zweiten Raumes

Über die grafische Oberfläche können von einem externen Computer also die Leuchtdioden des Joy-Pi beliebig gesteuert werden. Dabei wird der aktuelle LED-Status auf dem GUI angezeigt.

7.3 Verbindung

Die Verbindung zwischen Raspberry Pi (Server) und dem externen Computer (Client) wurde mit einer Netzwerkverbindung über TCP/IP realisiert. Die Daten, die dabei ausgetauscht werden, sind in Dictionaries abgespeichert. Im Quelltext 7.1 sind diese Dictionaries dargestellt. Auf der Client-Seite wird das Dictionary `räume` verwendet. Dabei werden den Schlüsselwörtern `R1` und `R2`, welche Raum 1 und Raum 2 darstellen, eine 0 zugewiesen, wenn ein Ausschalten seitens der grafischen Oberfläche erfolgt. Falls eingeschaltet wird, wird eine 1 zugeordnet. Ähnlich ist es bei dem Dictionary `LEDs_state` auf der Server-Seite: Hier wird der Status von *LED 1* und *LED 2* festgehalten. Eine ausgeschaltete Leuchtdiode bewirkt die Zuweisung einer 0, eine eingeschaltete

die Zuweisung einer 1. Im unten gezeigtem Quelltext 7.1 sind beide Räume ausgeschaltet und die LEDs leuchten nicht.

```
1 #Dictionary auf der Client-Seite
2 raeume = {"R1": 0, "R2": 0}
3
4 #Dictionary auf der Server-Seite
5 LEDs_state = {"LED1": 0, "LED2": 0}
```

Quelltext 7.1: Python-Code der verwendeten Dictionaries

Wird auf der grafischen Oberfläche z.B. *Raum 1* eingeschaltet, wird diese Information im Dictionary `raeume` abgespeichert und versendet. Sobald der Raspberry Pi den Datenstrom des Clients (externer Computer) ausgelesen hat, werden die Daten des Dictionaries `LEDs_state` aktualisiert und die Funktion zum Einschalten von *LED 1* ausgeführt. Da sich nun der Zustand der LED geändert hat, werden diese Daten zurück an den Client gesendet. Dieser aktualisiert wiederum sein Dictionary und führt die Funktion zum Einschalten der LED-Statusanzeige des ersten Raumes auf dem GUI aus.

Wird auf dem Joy-Pi nun *Taste 1* oder *Taste 2* gedrückt, werden entsprechende LED geschaltet und der Raspberry Pi sendet das aktuelle Dictionary `LEDs_state` zum externen Computer. Dieser wiederum liest diese Daten aus und zeigt den neuen LED-Status auf dem GUI an.

8 Zusammenfassung und Ausblick

Für dieses Projekt war ein Joy-Pi-Koffer gegeben, der ein Raspberry-Pi und viele Sensoren und Aktoren enthält. Mit diesem Koffer sollte eine angemessene Aufgabenstellung überlegt und umgesetzt werden.

Die Wahl fiel auf die Lichtsteuerung von zwei Räumen, die mit dem Joy-Pi, einem GUI und einer TCP/IP-Verbindung steuerbar sein sollte. Zuerst wurde die einfache Steuerung der LEDs mit den Tastern programmiert. Mit den Tastern konnte die zugehörige LED ein- und wieder ausgeschaltet werden. Als Nächstes wurde eine graphische Oberfläche entwickelt. Diese wurde zunächst auf dem Raspberry Pi aufgerufen, um ebenfalls die Lampen schalten zu können. Dabei gab das GUI über die LED-Statusanzeige eine Rückmeldung über den Zustand der Leuchten. Anschließend wurde durch eine TCP/IP-Verbindung ermöglicht, die grafische Oberfläche von einem anderen Computer aus aufzurufen und von dort aus die LEDs zu steuern. Währenddessen konnten die Leuchtdioden zusätzlich über die Hardware-Knöpfe des Joy-Pi-Koffers geschaltet werden. Dabei dient der Raspberry Pi als Server und der externe Computer als Client.

Nach Erstellung der Projektzielsetzung wurde der Joy-Pi-Koffer genauer betrachtet und es wurden über die einzelnen Komponenten des Koffers Informationen gewonnen. Zum Beispiel läuft der Raspberry Pi mit der Linux Version Raspbian. Außerdem ist auf diesem eine Programmierentwicklung vorinstalliert. Zu den Sensoren des Koffers gehören u.a. ein Berührungssensor, ein Ultraschallsensor und Taster. Als Aktoren befinden sich u.a. LEDs, ein Buzzer oder auch eine Siebensegment-Anzeige auf dem Joy-Pi. Für das Projekt waren die LEDs und die Taster relevant.

Die Realisierung erfolgte nun Schritt für Schritt gemäß Aufgabenstellung. Zunächst wurde ein Ein- und Ausschalten einer LED über einen Taster erreicht. Dabei wurden die Pins der benötigten LEDs und der Taster ausgewählt. Anschließend wurden in der Programmierung über die GPIO-Klasse alle benötigten Methoden aufgerufen, die notwendig sind, um die LED zum Leuchten zu bringen. Mit der Klasse Tkinter wurde als nächstes das GUI erstellt. Dadurch entsteht ein Fenster, womit die Steuerung der LEDs ausgeführt werden kann. Dabei werden den Schaltflächen die Funktionen für das Ein- und Ausschalten der Leuchtdioden zugeordnet. Als nächstes wurde die TCP/IP-Verbindung mit diesen beiden Steuerungen kombiniert. Dabei sollte der Raspberry Pi als Server eingerichtet werden und ein anderer Computer sich als Client mit diesem verbinden. Sobald beide verbunden waren, erfolgte ein Datenaustausch. Dabei wurden die Zustände der Leuchten an den Client und die Befehle, die über das GUI erfolgen, an den Server gesendet.

Das Ergebnis des Projektes ist eine funktionierende Lichtsteuerung für zwei Räume. Es kann hardwaremäßig über Taster und softwaremäßig über die grafische Oberfläche das Licht über eine TCP/IP-Verbindung gesteuert werden.

Diese Steuerung kann in vielen Bereichen eine Erweiterung finden. Zum Beispiel könnte sich die Schalthoheit aus der Schaltungstechnik zum Vorbild genommen werden. Dabei geht es um die Berechtigung aus der Ferne schalten zu dürfen. Dies könnte z.B. bei der Beleuchtung in einem Bürogebäude so umgesetzt werden, dass das Wachpersonal erst nach einer bestimmten Uhrzeit befugt ist, die Beleuchtung der Büros aus der Ferne zu schalten. Eine weitere Möglichkeit dieses Prinzip einzubringen, wäre in privaten Haushalten. Dabei würde die Schaltberechtigung aus der Ferne erst mit dem Verlassen der Wohnung aktiviert werden. So könnte ein missverständliches Ein- und Ausschalten der Beleuchtung durch eine Person im und einer Person außerhalb des Hauses ausgeschlossen werden. Weitere Erweiterungsmöglichkeit des Projektes gäbe es im Bereich des Smart Home. Dabei könnten mit Leichtigkeit weitere Räume zur Lichtsteuerung hinzugefügt

werden. Es könnte aber auch eine Steuerung von Rollläden, Herd und Garagentor ergänzt werden. So würde auch die grafische Oberfläche umfangreicher ausgestattet. Diese als Applikation auf dem Mobiltelefon darzustellen, könnte ebenfalls als eine mögliche Erweiterung angesehen werden. Falls der Eigentümer vergessen hat den Herd oder das Licht abzustellen, könnte dies unkompliziert über die Applikation auf dem Mobiltelefon nachgeholt werden.

Abbildungsverzeichnis

1.1	Übersicht der Antriebe	2
4.1	Zu enge Rohrdarstellung	7
4.2	Verwendung der Variable für die Sichtbarkeit	8
5.1	Der Joy-Pi-Koffer	9
5.2	Der Raspberry-Pi 3 B	10
5.3	Sensoren und Aktoren des Joy-Pi	10
5.4	Pin-Nummern und deren Bauteile	12
6.1	Im Projekt verwendete Tasten des Joy-Pi	13
6.2	Im Projekt verwendete LEDs des Joy-Pi	13
6.3	Grafische Oberfläche programmiert mit Python und Tkinter	15
6.4	Datenaustausch zwischen externem Computer und Joy-Pi	17
7.1	Projektaufbau mit genutzter Hardware	18
7.2	Zustände der LEDs	19
7.3	Grafische Oberfläche bei Betätigung der Einschalt-Taste des ersten Raumes	19
7.4	Grafische Oberfläche bei Betätigung der Ausschalt-Taste des ersten Raumes	20
7.5	Grafische Oberfläche bei Betätigung der Einschalt-Taste des zweiten Raumes	20
7.6	Grafische Oberfläche bei Betätigung der Ausschalt-Taste des zweiten Raumes	20

Tabellenverzeichnis

4.1	Tabelle zur Auflistung der Farben der Motoren	6
4.2	Tabelle zur Auflistung der Farben der Motoren	7
5.1	Tabelle zur Auflistung der Sensoren	11
5.2	Tabelle zur Auflistung der Aktoren	11

Quelltextverzeichnis

6.1	Python-Code zum Einlesen des Tastenzustandes	14
6.2	Python-Code zur Ansteuerung einer LED	15
6.3	Python-Code der verwendeten Dictionaries	15
7.1	Python-Code der verwendeten Dictionaries	21

Literaturverzeichnis

- [1] *Anleitung zum Joy-Pi.* anleitung.joy-it.net/wp-content/uploads/2018/10/RB-JoyPi-Anleitung.pdf. Eingesehen am 04.01.2020.
- [2] *Graphical User Interfaces with Tk.* <https://docs.python.org/3/library/tk.html>. Eingesehen am 14.12.2019.
- [3] *JSON - Introduction.* https://www.w3schools.com/js/js_json_intro.asp. Eingesehen am 18.12.2019.
- [4] *JSON encoder and decoder.* <https://docs.python.org/3/library/json.html>. Eingesehen am 17.12.2019.
- [5] Michael Kofler. *Python-Der Grundkurs.* 1. Auflage. Rheinwerk Computing, 2019. ISBN: 978-3-8362-6679-6. URL: <https://kofler.info/buecher/python/>.
- [6] *RPi.GPIO Project description.* <https://pypi.org/project/RPi.GPIO/>. Eingesehen am 13.12.2019.
- [7] *Send() Function Of Python Socket Class.* <https://pythontic.com/modules/socket/send>. Eingesehen am 02.01.2020.
- [8] *TCP/IP.* <https://www.elektronik-kompendium.de/sites/net/0606251.html>. Eingesehen am 17.12.2019.
- [9] *Technische Daten des Raspberry-Pi.* <https://www.elektronikpraxis.vogel.de/raspberry-pi-3-bringt-12-ghz-64-bit-quadcore-wlan-und-bluetooth-fuer-iot-a-523253/index2.html>. Eingesehen am 19.12.2019.
- [10] *Was ist ein Pullup- und ein Pulldown-Widerstand?* <https://www.elektronik-kompendium.de/public/schaerer/pullr.html>. Eingesehen am 18.12.2019.
- [11] *socket — Low-level networking interface.* <https://docs.python.org/3.8/library/socket.html>. Eingesehen am 02.01.2020.

A Anhang: Quelltext des Projektes

In diesem Kapitel werden die erstellten Quelltexte aufgeführt.

A1 Quelltext Main-Funktion des Servers

A2 Quelltext des Servers

A3 Quelltext Main-Funktion des Clients

A4 Quelltext des Clients

A.1 main_server.py

```
1 # -----
2 # Datei: main.py
3 # -----
4
5 # import-Anweisungen
6
7 # Konstantendeklaration
8 #from client import ClientGui
9 from server import TcpServer
10
11 #MY_IP_ADDR = "127.0.0.1"
12 MY_IP_ADDR = "10.3.47.157"
13 MY_PORT = 54321
14 MY_BYTES = 1024
15 MY_SLEEP = 1
16 MY_ANZ_VAL = 3
17
18
19 # -----
20 # main-Funktion fuer Programmeintritt
21 # -----
22 def main():
23     server = TcpServer(MY_IP_ADDR, MY_PORT, MY_BYTES, led1=33,
24                         button1=37, led2=29, button2=22)
25     #client = ClientGui(MY_IP_ADDR, MY_PORT, MY_BYTES)
26     #server.init_states()
27     #client.mainloop()
28
29
30 if __name__ == '__main__':
31     main()
32
```

A.2 server.py

```
1 import json
2 import socket
3 import threading
4
5 from RPi import GPIO
6
7
8 class TcpServer(threading.Thread):
9     def __init__(self, ip, port, byte_num, led1, led2, button1
10      , button2):
11         """
12             TcpServer fragt GPIOs ab und sendet Daten an TcpClient
13             :param ip:
14             :param port:
15             :param byte_num:
16             """
17         super().__init__()
18         self._v_sock = None
19         self.k_sock = None
20         self.__ip = ip
21         self.__port = port
22         self.__bytes = byte_num
23         GPIO.setmode(GPIO.BEAN)
24         self.led1_pin = led1
25         self.button1_pin = button1
26         self.led2_pin = led2
27         self.button2_pin = button2
28         #Festlegung der Ausgänge
29         GPIO.setup(self.led1_pin, GPIO.OUT)
30         GPIO.setup(self.led2_pin, GPIO.OUT)
31         #Festlegung der Eingänge
32         GPIO.setup(self.button1_pin, GPIO.IN, pull_up_down=GPIO
33             .PUD_DOWN)
34         GPIO.setup(self.button2_pin, GPIO.IN, pull_up_down=GPIO
35             .PUD_DOWN)
36         # Ereignis bei Betätigung des Buttons in Raum 1
37         GPIO.add_event_detect(self.button1_pin, GPIO.RISING,
38             callback=self.button1_interrupt, bouncetime=500)
39         # Ereignis bei Betätigung des Buttons in Raum 1
40         GPIO.add_event_detect(self.button2_pin, GPIO.RISING,
41             callback=self.button2_interrupt, bouncetime=500)
42
43     def init_states(self):
44
45         if GPIO.input(self.led1_pin) == GPIO.HIGH:
46             self.leds_state["led1"] = 0
47         else:
48             self.leds_state["led1"] = 1
```

```

46         if GPIO.input(self.led2_pin) == GPIO.HIGH:
47             self.leds_state["led2"] = 0
48         else:
49             self.leds_state["led2"] = 1
50         self._send()
51
52     def run(self):
53         """
54             Send current LED pin state to connected clients.
55
56             v_sock: Verbindungssocket
57             k_sock: Kommunikationssocket
58
59             :return: None
60             """
61         self.__v_sock = socket.socket(socket.AF_INET, socket.
62             SOCK_STREAM)
63         self.__v_sock.bind((self.__ip, self.__port))
64         self.__v_sock.listen(1)
65         try:
66             while True:
67                 self.k_sock, addr = self.__v_sock.accept()
68
69                 while True:
70                     data = self.k_sock.recv(self.__bytes)
71                     if len(data) > 0:
72                         print("Server data: {}".format(data.
73                             decode())))
74                         data_json = json.loads(data.decode())
75                         if 'init' in data_json.keys():
76                             #Schlüssel (Element) im dictionary?
77                             self.init_states()
78                         else:
79                             self.leds_state["led1"] = data_json
80                             self.leds_state["led2"] = data_json
81                             [ "R1" ]
82                             self.change_led()
83                             # self.k_sock.send(json.dumps(
84                             data_dict).encode())
85                             else:
86                             self.k_sock.close()
87                             break
88
89                         finally:
90                             self.__v_sock.close()
91
92             def change_led(self):
93                 if self.leds_state["led1"] == 1:
94                     self.licht_an(self.led1_pin)
95                 if self.leds_state["led2"] == 1:

```

```
91         self.licht_an(self.led2_pin)
92     if self.leds_state["led1"] == 0:
93         self.licht_aus(self.led1_pin)
94     if self.leds_state["led2"] == 0:
95         self.licht_aus(self.led2_pin)
96
97     def licht_an(self, pin):
98         """
99             Turn LED on
100            :param pin: RPi board pin number where LED is
101            connected to.
102            """
103        if GPIO.input(pin) == GPIO.HIGH:
104            GPIO.output(pin, GPIO.LOW)
105            print("Licht ist bei Pin {} an".format(pin))
106            self.__send()
107
108    def licht_aus(self, pin):
109        if GPIO.input(pin) == GPIO.LOW:
110            GPIO.output(pin, GPIO.HIGH)
111            print("Licht ist bei Pin {} aus".format(pin))
112            self.__send()
113
114 # Interrupt- Funktion für den Button in Raum 1
115 def button1_interrupt(self, channel):
116     if GPIO.input(self.led1_pin) == GPIO.HIGH:
117         self.leds_state["led1"] = 1
118         self.licht_an(self.led1_pin)
119     else:
120         self.leds_state["led1"] = 0
121         self.licht_aus(self.led1_pin)
122
123 def button2_interrupt(self, channel):
124     if GPIO.input(self.led2_pin) == GPIO.HIGH:
125         self.leds_state["led2"] = 1
126         self.licht_an(self.led2_pin)
127     else:
128         self.leds_state["led2"] = 0
129         self.licht_aus(self.led2_pin)
130
131 def __send(self):
132     msg = json.dumps(self.leds_state)
133     self.k_sock.send(msg.encode())
134
```

A.3 main_client.py

```
1 # -----
2 # Datei: main.py
3 # -----
4
5 # import-Anweisungen
6
7 # Konstantendeklaration
8 from client import ClientGui
9
10 MY_IP_ADDR = "10.3.47.157"
11 MY_PORT = 54321
12 MY_BYTES = 1024
13 MY_SLEEP = 1
14 MY_ANZ_VAL = 3
15
16
17 # -----
18 # main-Funktion fuer Programmeintritt
19 #
20 def main():
21     #server = TcpServer(MY_IP_ADDR, MY_PORT, MY_BYTES, Led1=33
22     , button1=37, led2=29, button2=22)
22     #server.start()
23     client = ClientGui(MY_IP_ADDR, MY_PORT, MY_BYTES)
24     #server.init_states()
25     client.mainloop()
26
27
28 if __name__ == '__main__':
29     main()
30
```

A.4 client.py

```

1 import json
2 import socket
3 import threading
4 import tkinter as tk
5
6
7 class TcpClient(threading.Thread):
8     def __init__(self, gui, ip, port, byte_num):
9         super().__init__()
10        self.__gui = gui
11        self.k_sock = None
12        self.__ip = ip
13        self.__port = port
14        self.__bytes = byte_num
15
16    def run(self):
17        self.k_sock = socket.socket(socket.AF_INET, socket.
18            SOCK_STREAM)
19        self.k_sock.connect((self.__ip, self.__port))
20        try:
21            while True:
22                data = self.k_sock.recv(self.__bytes)
23                data_dict = json.loads(data.decode())
24                print("Client data: {}".format(data_dict))
25                self.__gui.display_led_state(data_dict["led1"]
26                ], data_dict["led2"])
27        finally:
28            self.k_sock.close()
29
30 class ClientGui(tk.Tk):
31     def __init__(self, ip, port, byte_num):
32         super().__init__()
33         # Übergabe des TCP Clients
34         self.__tcp_client = TcpClient(self, ip, port, byte_num)
35         self.__tcp_client.start()
36         # Titel und Maße des Fensters
37         self.title("Lichtsteuerung")
38         self.geometry("550x80")
39         # Frame für Raum 1 wird generiert
40         self.lbl = tk.LabelFrame(self, text="Raum 1", width=250
41             , height=80)
42         self.lbl.grid(row=0, column=0, columnspan=2, sticky="W"
43             , padx=5, pady=0, ipadx=0, ipady=0)
44         # Einschalt-Button für Raum 1
45         self.an_btn = tk.Button(self.lbl, text="Einschalten",
46             command=self.cb_an_r1)
47         self.an_btn.grid(column=0, row=1, padx=15, pady=10)
48         # Ausschalt-Button für Raum 1

```

```
46         self.aus_btn = tk.Button(self.lbl, text="Ausschalten",
47             command=self.cb_aus_r1)
48         self.aus_btn.grid(column=1, row=1, padx=15, pady=10)
49         # LED-Anzeige für Raum 1
50         self.anzeige1 = tk.Button(self.lbl, text="      ", bg="grey",
51             command="")
52         self.anzeige1.grid(column=2, row=1, padx=15, pady=10)
53         # Frame für Raum 2 wird generiert
54         self.lbl2 = tk.LabelFrame(self, text="Raum 2", width=
55             250, height=80)
56         self.lbl2.grid(row=0, column=2, columnspan=2, sticky="W",
57             padx=5, pady=0, ipadx=0, ipady=0)
58         # Einschalt-Button für Raum 2
59         self.an_btn = tk.Button(self.lbl2, text="Einschalten",
60             command=self.cb_an_r2)
61         self.an_btn.grid(column=0, row=1, padx=15, pady=10)
62         # Ausschalt-Button für Raum 2
63         self.aus_btn = tk.Button(self.lbl2, text="Ausschalten",
64             command=self.cb_aus_r2)
65         self.aus_btn.grid(column=1, row=1, padx=15, pady=10)
66         # LED-Anzeige für Raum 2
67         self.anzeige2 = tk.Button(self.lbl2, text="      ", bg="grey",
68             command="")
69         self.anzeige2.grid(column=2, row=1, padx=15, pady=10)
70         self.raeume = {"R1": 0, "R2": 0}
71         self.__tcp_client.k_sock.send('{"init": 1}'.encode())
72
73     def __send(self):
74         msg = json.dumps(self.raeume)
75         self.__tcp_client.k_sock.send(msg.encode())
76
77     def display_led_state(self, led1_state, led2_state):
78         """
79             Set displayed LED states
80             :param led1_state: State of LED1
81             :param led2_state: State of LED2
82         """
83         try:
84             if led1_state and not led2_state:
85                 print("Client macht LED1 gelb und LED2 grau")
86                 self.anzeige1.configure(bg="yellow")
87                 self.anzeige2.configure(bg="grey")
88                 self.raeume["R1"] = 1
89                 self.raeume["R2"] = 0
90             elif not led1_state and led2_state:
91                 print("Client macht LED2 gelb und LED1 grau")
92                 self.anzeige1.configure(bg="grey")
93                 self.anzeige2.configure(bg="yellow")
94                 self.raeume["R1"] = 0
```

```
89         self.raeume["R2"] = 1
90     elif led1_state and led2_state:
91         print("Client macht LED2 gelb und LED1 gelb")
92         self.anzeige1.configure(bg="yellow")
93         self.anzeige2.configure(bg="yellow")
94         self.raeume["R1"] = 1
95         self.raeume["R2"] = 1
96     else:
97         print("Client macht LED2 grau und LED1 grau")
98         self.anzeige1.configure(bg="grey")
99         self.anzeige2.configure(bg="grey")
100        self.raeume["R1"] = 0
101        self.raeume["R2"] = 0
102    except Exception as e:
103        print(e)
104
105    # Callback_Funktion um LED für Raum 1 einzuschalten
106    def cb_an_r1(self):
107        self.raeume["R1"] = 1
108        self.__send()
109
110    # Callback_Funktion um LED für Raum 1 auszuschalten
111    def cb_aus_r1(self):
112        self.raeume["R1"] = 0
113        self.__send()
114
115    # Callback_Funktion um LED für Raum 2 einzuschalten
116    def cb_an_r2(self):
117        self.raeume["R2"] = 1
118        self.__send()
119
120    # Callback_Funktion um LED für Raum 2 auszuschalten
121    def cb_aus_r2(self):
122        self.raeume["R2"] = 0
123        self.__send()
124
```