Hochschule für Technik und
Wirtschaft Berlin

Independent Coursework

Erstellung eines interaktiven Informationssystems auf Grundlage eines LED Panels

Fabian Bänsch

betreut von
Prof. Dr. Tobias Lenz

4. Februar 2018

Inhaltsverzeichnis

1 Vorstellung

Diese Arbeit soll zeigen, wie mit relativ günstigen Komponenten ein Informationshub gebaut werden kann. Primär sollen hier Echtzeitinformation über eine Anzeigetafel erscheinen.

Die Idee dieser Arbeit entstand dabei schon vor langer Zeit. Ich hatte mir dazu aus einem Onlineshop für Industriebedarf eine günstige LED Tafel besorgt. Diese werden normalerweise zu hunderten miteinander verbunden, sodass große Leinwände für Public Viewings entstehen. Für meine Absichten sollte aber erst einmal eines solcher Module reichen.

2 Konzept

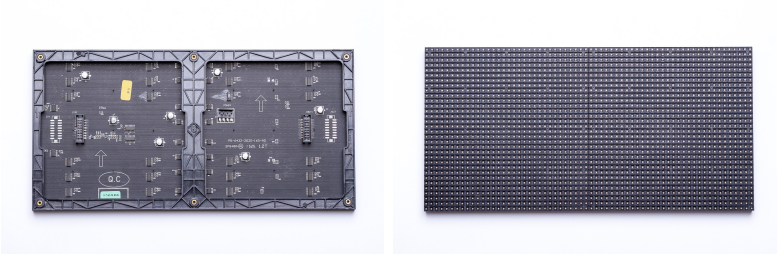
Das Konzept der Arbeit soll im Allgemeinen die Interaktivität in Wohngemeinschaften steigern. Als zentrales Element in Küchen soll so ein Mehrwert für die Bewohner entstehen. Zum einen sollen Anwendungen entstehen, welche unidirektional Informationen liefern. So kann das aktuelle Wetter, die Abfahrtzeiten der U-Bahn vor der Haustür oder aber interessante Nachrichten aus der Umgebung angezeigt werden. Für die bidirektionalen Informationsaustausch soll zudem eine Art Pinnwand entwickelt werden, worauf die Benutzter Nachrichten hinterlassen können. Zuletzt soll auch die Interaktivität mittels eines kleinen Spiels gesteigert werden.

Für die technische Konzeptionen soll das LED Panel mit einem kleinen Rechner verbunden werden, welcher sämtliche Logik steuert. Das LED Panel ist dafür nicht geeignet. Für die Steuerung des Panels soll zudem eine Smartphone App erstellt werden, welche neben der Anpassung der Einstellungen auch Möglichkeiten bietet Nachrichten auf die Pinnwand zu schreiben oder als Eingabegerät für Spiele zu dienen.

Um die Energiekosten für den Informationshub nicht in die Höhe zu treiben, soll zudem eine Bewegungserkennung integriert werden. Somit muss das Gerät nicht in der Nacht weiterlaufen.

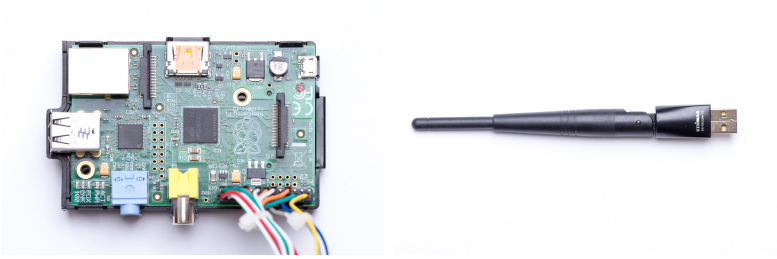
3 Hardware

Dieses Projekt ist teilweise sehr Hardwarelastig. Daher folgt nun eine kleine Zusammenstellung der wichtigsten Komponenten. Insgesamt beliefen sich die Kosten dabei auf ca. 60 Euro.



(a) LED Panel Rückseite **(b)** LED Panel Vorderseite

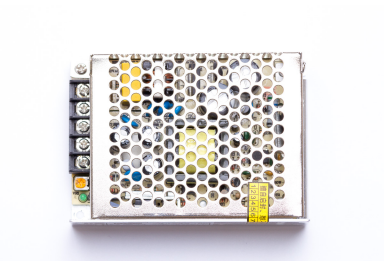
Abb. 1
LED Panel, 64 mal 32 RGB LEDs, Abmessungen - 32 mal 16 cm. Anschluss via einem HUB 75 Port. Stromversorgung - 5 V DC.



(a) Raspberry PI **(b)** WLAN Modul

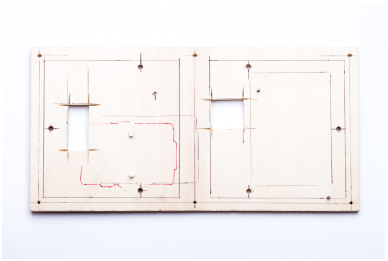
Abb. 2
Raspberry PI 1 Model B (700 MHz und 512 MB RAM) in Kobinationen mit einem WLAN Modul für den Internetzugang.

(a) Bewegungsmelder



(b) Stromwandler

Abb. 3
Infrarot Bewegungsmelder und Stromwandler von AC auf 5 V DC bei 10 A.



(a) Holz

(b) Schrauben

Abb. 4
Weiteres Zubehör, wie Holz als Basiselement der Konstruktion sowie Schrauben für die Befestigung.

3.1 Inbetriebnahme

Die spätere Kombination von Modulen soll am Ende wie in Abbildung 5 aussehen. Das LED Panel wird durch den HUB 75 Anschluss via Jumper Kabel an die GPIO Pins des Raspberrys angeschlossen. Ein enstprechende Ansteuerung findet im Raspberry statt. Daneben wird der IR Bewegungssensor an die GPIO Pins angeschlossen. Die Stromversorgung des LED Panels wird via 5 V Gleichstrom durch den Stromwandler bewältigt. Das WLAN Modul kann einfach per USB an den Raspberry angeschlossen werden.

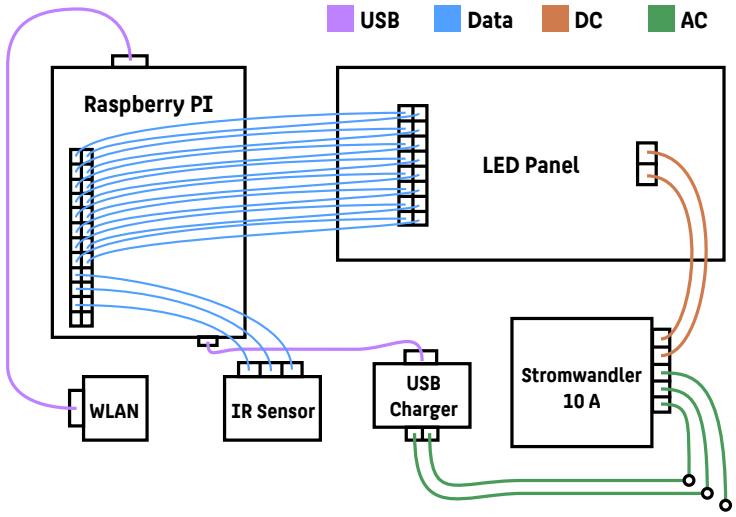


Abb. 5
Schematische Darstellung der relevanten Hardwarekomponenten und deren Verbindung.

3.2 Framework von Hzeller

Auf der Suche nach bereits vorhanden Frameworks für die Ansteuerung von LED Panels durch einen seriellen HUB 75 Port bin ich auf das Framwork von Henner Zeller gestoßen [1]. Dort befindet sich zudem eine Dokumentation für die Belegung des HUB 75 Anschlusses und wie diese auf die GPIO Pins des Raspberry gemappt werden können, um später das Framework nutzen zu können. Das Framework selbst wird im Kapitel 5 weiter erläutert.

4 Anwendungsdesign

Das Konzepte kann in drei größere Unterkategorien unterteilt werden. Auf dem Raspberry selbst soll es ein Softwaremodul geben, welche die Steuerung des LED Panels durch das eben vorgestellte Framework regelt. Zum Anderen können hier auch Untermodule hinzugefügt werden. Diese kleinen Module dienen später den verschiedenen Anzeigemöglichkeiten (z.B. Wetterfunktionalität etc.). Für die interaktive Steurung soll es zudem eine Smartphone App geben. Diese kann mittels des Internets oder im lokalen Netzwerk mit dem Raspberry kommunizieren. Um diese Kommunkation mit dem Internet zu vereinfachen, soll als Schnittstelle eine eigene API errichtert werden. Eine Veranschaulichung des Zusammenspiels der verschiededenen Komponenten befindet sich in Abbildung 6.

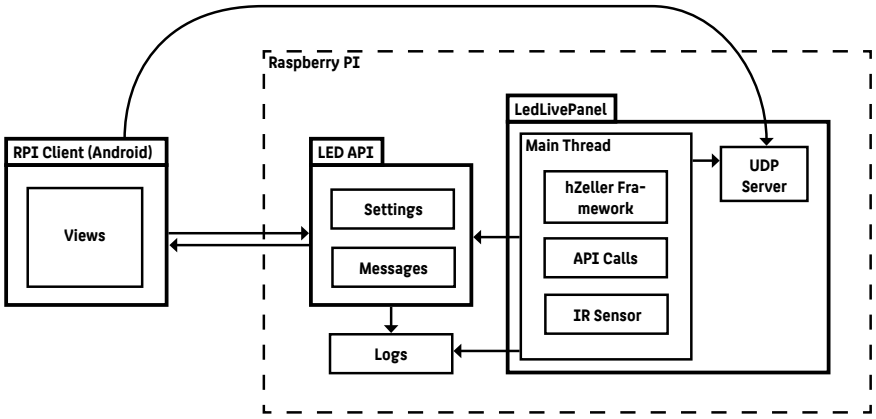


Abb. 6
Schematische Darstellung der relevanten Softwarekomponenten und deren Abhängigkeiten.

5 LED Python Modul

Das Kernstück der Arbeit liegt in der Anwendung für die Steuerung des LED Panels. Diese umfasst dabei zum einen die Nutzung der API des Frameworks [1] sowie die Möglichkeit Informationen aus dem Internet zu erhalten. Zudem soll sie Informationen aus des IR Bewegungssensor verarbeiten und

auch auf dynamische Einstellungen seitens der Benutzer reagieren können.

Als Programmiersprache wurde auf Python gesetzt. Diese ist schnell zu erlernen und hat viele Ähnlichkeiten zu anderen Sprachen. Zudem ist die Ansteuerung des Framworks ebenfalls durch Python möglich. Als zentraler Programmstart dient die Klasse `LedLivePanel`.

5.1 Ansteuerung des LED Panels

Das Framework kommt sogleich mit mitgelieferten Beispielanwendungen. Diese können gestartet werden, um die Anbindung des Raspberrys mit den LED Panel zu testen. Im Falle eines 64*32 LED Panels kann ein Test mit folgender Eingabe gestartet werden:

```
sudo python pulsing-colors.py -r 32 -c 2
```

Ein Blick in die Implementierung zeigt dabei die Einfachheit des Frameworks. Zunächst muss die Klasse für die Programmlogik von der Klasse `Samplebase` erben. Damit stehen schon alle wichtigen Datenstrukturen und Anbindungen zur Hardware bereit. Beim überschreiben der `run()` Methode wird dabei dieser Abschnitt immer wiederkehrend aufgerufen. Dies entspricht somit einer „zeichnen" Methoden mit automatisierten Intervall zum Aufrufen. Für die eigentliche Steuerung des LED Panels können von der Superklasse verschiedene Methoden aufgerufen werden.

```
graphics.DrawText(canvas, font, x, y, color, 'Text')
```

Hiermit kann ein Zeichenkette mit einer bestimmten Farbe und Form an einer Position auf das Canvas gezeichnet werden.

```
canvas.Clear()
```

Diese Methode leert das Canvas.

```
canvas.SetPixel(x, y, 255, 255, 255)
```

Mit dieser Funktion lassen sich einzelne Pixel auf einen bestimmten Farbwert setzen. Mithilfe dieser wenigen Methoden lassen sich schon viele, wenn nicht sogar alle Anwendungsfälle abdecken. Es wurden für die Wiederverwendbarkeit noch eigene Hilfsfunktionen erarbeitet. So können auch lange Zeichenketten automatisiert mit Zeilenumbrüchen und einem Scrolling Verhalten

auf dem LED Panel angezeigt werden. Ebenso gibt es Funktionalitäten, um ganze Bilder auf das Canvas zu projizieren.

5.2 Echtzeitinformationen

Im weiteren Verlauf wird dargestellt, wie Echtzeitinformationen erhalten, verarbeitet und angezeigt werden können. Im Wesentlichen steht hier die Klasse `API_Calls` im Vordergrund. Sie dient der Kontaktaufnahme mit externen APIs.

5.2.1 Beispiel - New York Times

Als einfache Variante soll kurz anhand der New York Times API [2] beschrieben werden, wie mit Python HTTP Get Methoden ausgeführt und verarbeitet werden können. Dazu wird im wesentlichen das Python Modul `requests` verwendet. Hier veranschaulicht ist die API von "http://www.nytimes.com/". Die Authentifizierung erfolgt mittels eines Query-Parameters api-key. Dieser kann von der Seite nach einer Registrierung angefordert werden. Sämtliche Schlüssel und sicherheitsrelevante Daten sind in der `Settings` Klasse gespeichert. Diese ist natürlich nicht für die Öffentlichkeit bestimmt und ist dementsprechend auch nicht im abgegeben Source Code enthalten.

```
apiURL = Settings.nyt_api+'?limit='+
         str(led_settings.nyt_num_of_news)+
         '&api-key='+Settings.nyt_api_key
news_req = requests.get(apiURL, timeout=10)
news = news_req.json()
```

Hier wurde der gesamte Pfad per Hand gebaut. Es ist jedoch auch möglich die einzelnen Query-Parameter einzelnen zu deklarieren und später der `get()` Methode zu übergeben. Der zurückgelieferte Inhalt wird mit der `json()` Methode in eine objekt-orientiertes gebräuchliches Format konvertiert. Im Folgenden kann mittels einfacher Array Navigation das Json Objekt ausgewertet werden. Die Informationen werden dann im Anschluss zu einer großen Zeichenkette zusammengefasst, welche dann auf dem LED Panel angezeigt werden kann.

```
results = news['results']
for result in results:
        title = result['title'].encode('utf-8')
        abstract = result['abstract'].encode('utf-8')
```

5.2.2 Weitere Untermodule

Es sind noch eine Vielzahl weitere Informationsquellen enthalten. Diese werden nur kurz angerissen und deren Funktionen und Besonderheiten vorgestellt. Beispiele für deren spätere Verwendung können im Kapitel Impressionen eingesehen werden.

- **Pinnwand**

  Mittels einer eigens entwickelten API können hier interaktiv Nachrichten auf das LED Panel gepostet werden. Dieses Untermodul ruft dabei die letzten Nachrichten ab und zeigt sie an. Die API wird später vorgestellt (Kapitel 6).

- **Zitat des Tages**

  Hier wird das Zitat des Tages [3] abgerufen und angezeigt.

- **Twitter**

  In diesem Modul werden die letzten Tweets der Twitter API [4] angezeigt. Diese werden nach Hashtag gefiltert. Der Hashtag kann, wie später gezeigt, durch die Smartphone App verändert werden. Die API wird mit Twython [5] angesteuert.

- **Instagram**

  Ähnlich dem Twitter Modul werden hier durch die Angabe eines Hashtags Bilder aus der Instagram API [6] angezeigt.[1] Dort werden die Instagramtypischen quadratischen Bilder mit einem Scroll Effekt über das LED Panel angezeigt.

[1] Die API hat die Authentifizierung von OAuth nach OAuth2 geändert. Derzeit ist dieses Modul daher nicht mehr funktionsfähig.

- **Wetter**

  Aktuelle Informationen zum Wetter und Vorhersagen für den Tag werden durch die API von OpenWeatherMap [7] bereitgestellt.

- **Zeit Online**

  Als deutschsprachiges Nachrichtenportal wurde Zeit Online gewählt. Hier werden mittels der API [8] jeweils die neusten Artikel geladen und auf dem LED Panel angezeigt. Dazu zählt der Titel und das Abstract der Nachricht.

- **Tag des ...**

  Die Website "Welcher Tag ist Heute" [9] bietet interessante Informationen zum jeweiligen Tag an. Zum Beispiel der 12. November ist Tag der schlechten Wortspiele. Da der Anbieter keine API anbieten, wurde ein Website Scrapper geschrieben. Dieser basiert auf Grundlage von `BeautifulSoup` [10].

- **VBB**

  Der Verkehrsverbund Berlin-Brandenburg bietet eine eigene API [11] an, welche über Ankunfts- und Abfahrzeiten öffentlicher Verkehrsmittel Auskunft gibt. Interessant sind dabei die Zeiten von der Station vor der Haustür. In diesem Fall werden auf dem LED Panel, ähnlich wie auf den Fahrtenanzeiger auf Bahnhöfen, die nächsten U-Bahnen und die Restminuten bis zur Abfahrt angezeigt.

- **TXL Anküfte & Abflüge**

  Kostenfreie APIs für Statusinformationen von Flughäfen sind nach ausgiebiger Recherche nicht vorhanden. Daher wurde kurzerhand ebenfalls ein Website-Scrapper gebaut. Dieser liefert die Daten von der offiziellen Website der Berliner Flughäfen. Da der Scrapper relativ komplex gestaltet ist, befindet sich dieser in einer extra Klasse (`TXL_Scraper`).

5.3 Snake & UDP Server

Desweiteren wurde, um die Interaktivität zu steigern, ein Spiel für das LED Panel entwickelt. Ich habe mich dabei für die Umsetzung des klassischen Snake entschieden. Dieses ist aufgrund der historischen Pixel Raster Grafik bestens dafür geeignet. Die Spiellogik kann im Source Code begutachtet werden. Eine Beschreibung findet hier nicht statt.

Viel interessanter ist die Steuerung des Spiels. Da das LED Panel an der Wand hängen soll ohne weitere Peripherie Geräte, ist eine Eingabe durch Tastatur, Maus oder Gamecontroller ausgeschlossen. In der Konzeptionsphase wurde bereits erwähnt, dass es eine Smartphone App geben soll. Diese kann dann auch, wie später beschrieben, einen eigenen Gamecontroller besitzen.

Als Übertragungsprotokoll von Smartphone zum Raspberry wurde UDP eingesetzt. Dieses ist schnell, einfach zu implementieren und für diese Nutzung gut geeignet. Zum Empfang von UDP Paketen wird daher ein UDP-Server benötigt. Dieser wird in einem extra Thread gestartet, um die eigentliche Anwendung nicht zu unterbrechen. Fortlaufend werden dort dann entsprechende Pakete angenommen und als Eingabe gespeichert.

Das Snake Spiel kann darauf aufbauend einfach das zuletzt empfangene Paket abrufen und eine entsprechende Richtungsänderung erwirken. Der UDP-Server wurde in der Klasse `Udp_Key_Input` implementiert und zum Start der Anwendung asynchron gestartet.

5.4 IR Bewegungsmelder

Die bisherige Implementation hat bis dahin noch keine Logik, wann sie starten und stoppen soll. Dafür soll der IR-Bewegungsmelder sorgen. Daher wurde die bisherige Logik gekapselt und soll nun via des Sensors gestartet werden.

Der Sensor verfügt für die Kommunikation einen Datenausgangskanal, welcher mit einem Jumper Kabel an einen der freien GPIO Pins angeschlossen werden kann. Das Auslesen des Kanals erfolgt mit der GPIO Python Libary [13]. Die vereinfachte Form sei mittels des folgenden Code Beispiels verdeutlicht.

```
GPIO.setmode(GPIO.BOARD)
GPIO_PIR = 10
GPIO.setup(GPIO_PIR,GPIO.IN)
current_state = GPIO.input(GPIO_PIR)
```

Sollte dabei der `current_state` den Wert 1 annehmen, so wurde eine Bewegung erkannt. Der Wert 0 steht für keine Bewegung. Die Sensibilität und Reichweite des Sensors kann dabei mit 2 Stellschrauben verändert werden.

Zur Laufzeit des Programms erfolgt dann ein einfaches Polling auf diesen aktualisierten Wert und bei Bedarf wird die vorher gekapselte Logik ausgeführt. Das heißt es wird eine Schleife über alle vorgestellten Untermodule vollführt, wobei mittels Abfrage der Settings einzelne Untermodule auch übersprungen bzw. mit weiteren Information angereichert werden.

6 LED API

Wie bereits in der Konzeption beschrieben, soll es auf dem Raspberry eine eigene API geben, um einerseits das LED Panel mit der Außenwelt steuerbar zu machen und andererseits wichtige Daten permanent zu speichern. Hierzu wurde das Python Micorframework Flask [14] benutzt. Dieses leichtgewichtige Framework, basierend auf Werkzeug [15], erlaubt eine schnelle und einfache API Erstellung. Es sollen dabei folgende Aspekte mit abgedeckt werden.

- Speicherung von Benutzernachrichten
- Annehmen und Ausgeben dieser Nachrichten
- Speicherung von Einstellungen für den Betrieb des Panels
- Annehmen neuer und Ausgabe bisheriger Einstellungen
- Ausgabe einer Versionsnummer
- Ausgabe von Log Einträgen

Die entsprechenden Routen für die API sehen dabei wie folgt aus:

GET	/messages
GET	/messages/<int:message_id>
POST	/messages
DELETE	/messages/<int:message_id>
GET	/messages/newest
GET	/messages/newest/<int:num_messages>
GET	/settings
POST	/settings
GET	/logs/led/<int:num_of_logs>
GET	/logs/api/<int:num_of_logs>
GET	/version

6.1 Messaging

Um das LED Panel mit Nachrichten der Bewohner zu versorgen, wurden wie eingangs die entsprechenden Routen definiert. Eine Nachricht besteht dabei aus dem Autor, dem Inhalt und dem Zeitstempel. Die Speicherung dieser Daten soll lokal auf dem Gerät im Filesystem gespeichert werden. Von der Verwendung einer Datenbank wird aus Vereinfachung abgesehen. Die Datei mit allen Nachrichten soll dabei als JSon Datei vorliegen. Dies vereinfacht später das Ein- und Auslesen.

Das Routung durch Flask erfolgt mittels Annotationen an den Methoden. Hier wird zusätzlich angegeben, welche HTTP Methoden dazu wirken sollen. Eine zusätzliche Annotation sichert zudem die API mit der OAuth Authentifizierung. Die Login Informationen sind dabei statisch durch ein Key Value Paar in der API verankert.

In dem hier gezeigten Code Beispiel werden dabei alle Nachrichten durch die API zurückgegeben. Dazu wird die JSon Datei eingelesen, verändert und zurückgegeben. Die Objekte wurden dahingehend geändert, dass eine URI hinzugefügt wurde, um später in der Anwendung die Verarbeitung zu vereinfachen.

```
@app.route('/led/api/v1.0/messages', methods=['GET'])
@auth.login_required
def get_messages():
    messages = loadJson('messages.json')
    return jsonify({'messages':
    [make_public(message) for message in messages]})
```

Es wurden noch weitere Routen für das Nachrichtensystem implementiert. Wie üblich Methoden zum Hinzufügen neuer Nachrichten und zum Löschen bestehender Nachrichten. Zudem wurden weitere Methoden hinzugefügt, welche nur die neusten Nachrichten mit einer definierten Anzahl zurückgeben. Die API ist daher nicht konsequent Restful.

6.2 Weitere Routen

Es wurden neben der API für Benutzernachrichten auch weitere Routen hinzugefügt. Die einfache Route zur Version wird dabei genutzt, um Clients zu informieren, dass sie eine nicht mehr kompatible Variante der API benutzen. Die Route zu den Einstellungen wird genutzt, um benutzerdefinierte Konfigurationen am LED Modul zu persistieren. Das LED Modul kann daruf aufbauen seine Logik anpassen (z.B. bentzerdefinierte Hashtags für Twitter oder das Abschalten bestimmter Untermodule).

7 Android App

Als weiteres Kernstück der Arbeit wurde eine Smartphone App erstellt, welches eine Interaktion mit dem LED Modul und dem Panel erlaubt. Diese setzt dabei auf die bereits erläuterten Konzepte auf. Das heißt, sie benutzt sowohl die eben vorgestellte API als auch den UDP Server.

Die beim Start angezeigte View listet dabei alle bereits in der API enthaltenen Nachrichten auf. Via dem Floating Action Button (unten rechts) kann man eine neue Nachricht verfassen. Der Author kann dabei auch lokal auf dem Gerät gespeichert werden. Im Menü sind weitere Views erreichbar. Unteranderem die Ansicht zum Editieren der Einstellungen. Hier wird zunächst die aktuelle Konfiguration des LED Moduls geladen. Der Benutzer

kann anschließend via Switch Buttons oder Texteingabefelder das Verhalten steuern. Als Debugging Feature wurde zudem die Möglichkeit der Einsicht von Log Datein des Raspberrys eingerichtet (Logs des LED Moduls und der API). Diese Logs werden ebenfalls von der API bereitgestellt.

Um das Snake Spiel bedienen zu können, wurde ein UDP Client in die App integriert. Da es nicht sinnvoll wäre, den Übertragungsweg über das Internet zu routen (längere Verzögungszeit), findet der Datenaustausch innerhalb des Netzwerks statt. Dabei kann die Schlange entweder via Buttons oder durch Swipe-Bewegungen gesteuert werden. Dies gelingt nach einigen Tests sehr stabil und schnell.

Eine obligatorische View für lokale Einstellungen ist ebenfalls enthalten. Hier kann der Benutzer seinen Namen permanent setzten. Zudem kann hier der Fully Qualified Domain Name inklusive des Ports für die API eingestellt werden. Gleiches gilt für den UDP Server.

8 Weiteres

Für den produktiven und dauerhaften Einsatz mussten zudem noch weiteren Maßnahmen getroffen werden. Da im seltensten Falls der Raspberrry an einem Router mit statische IP zum Internet hängt, muss eine dynamsche DNS eingerichtet werden. Ich habe mich dabei für den Dienstleister No-Ip [16] entschieden. Bei anderen Anbieter ist das Verfahren jedoch ähnlich. Nach der Anmeldung erhält man die Zugansdaten für sein DNS. Diese muss nun dahingehend verändert werden, dass die Adresse auf die sich ständig veränderde Addresse des Routers zeigt. Dies gelingt, indem der Raspberry ab und zu den dynamischen DNS Server anpingt mit den Zugangsinformation. Der Server wird darauf hin die IP Adresse in der DNS Tabelle ändern (Mit den Werten aus den Ping Paket). Somit ist der Router ständig via einer Domain erreichbar.

Um jedoch den Raspberry hinter dem Router zu erreichen muss ein Portforwarding im Router eingestellt werden. In dieser Anwenung erfolgt dies mit dem Port 5000.

Mit diesen Setup ist sowohl der Raspberry als auch die App bereits voll funktionstüchtig. Sollte es aber aus unerklärlichen Gründen zum Neustart

des Raspberrys kommen, so wird zwar das Betriebssystem geladen aber keine Anwenung gestartet. Mit Cronjobs können solche Aufgaben erledigt werden. Dazu wurden zunächst 2 Shell Scripts geschrieben. Sie navigieren durch die Verzeichnisse und führen die Python Programme aus.

```
cd /home/pi/disp/led-api
python app.py
cd /
```

Diese Skripte wurden nun dem Crontab hinzugefügt. Die entsprechenden Zeilen sehen dabei wie im unteren Listing aus.

```
@reboot sh /home/pi/disp/led_api_launcher.sh
        >/home/pi/disp/logs/api_cronlog 2>&1
@reboot sh /home/pi/disp/rpi_led_launcher.sh
        >/home/pi/disp/logs/rpi_led_cronlog 2>&1
0 4 * * * sudo reboot
```

Die Kurzschreibweise `@reboot` steht für die Ausführung nach erfolgreichem Neustart. Die Zahlenkombination `0 4 * * *` steht für eine automatisierte Ausführung an jedem Tag um 4 Uhr Nachts. Hierbei wird der Raspberry neugestartet.

9 Fazit

In dieser Arbeit wurde gezeigt, wie man mit relativ simpler Hardware ein vollfunktionsfähiges Echtzeit-Informationssystem bauen kann. Dazu wurde zunächst ein Überblick über das Zusammenspiel verschiedener Hardware Komponenten und das Wirken der einzelnen Softwaremodule erläutert. Anschließend wurde ein Einblick in die Ansteuerung des LED Panels durch ein Raspberry gegeben und wie verschiedendste Information aus bereits bestehenden APIs zusammengetragen und visualisiert werden können. Ein kleiner Einblick in die eigene Erstellung einer API und das Nutzen von Smartphones zur Interaktion haben die Arbeit dabei abgerundet und in einen interaktiven und sinnvolen Kontext gestellt.

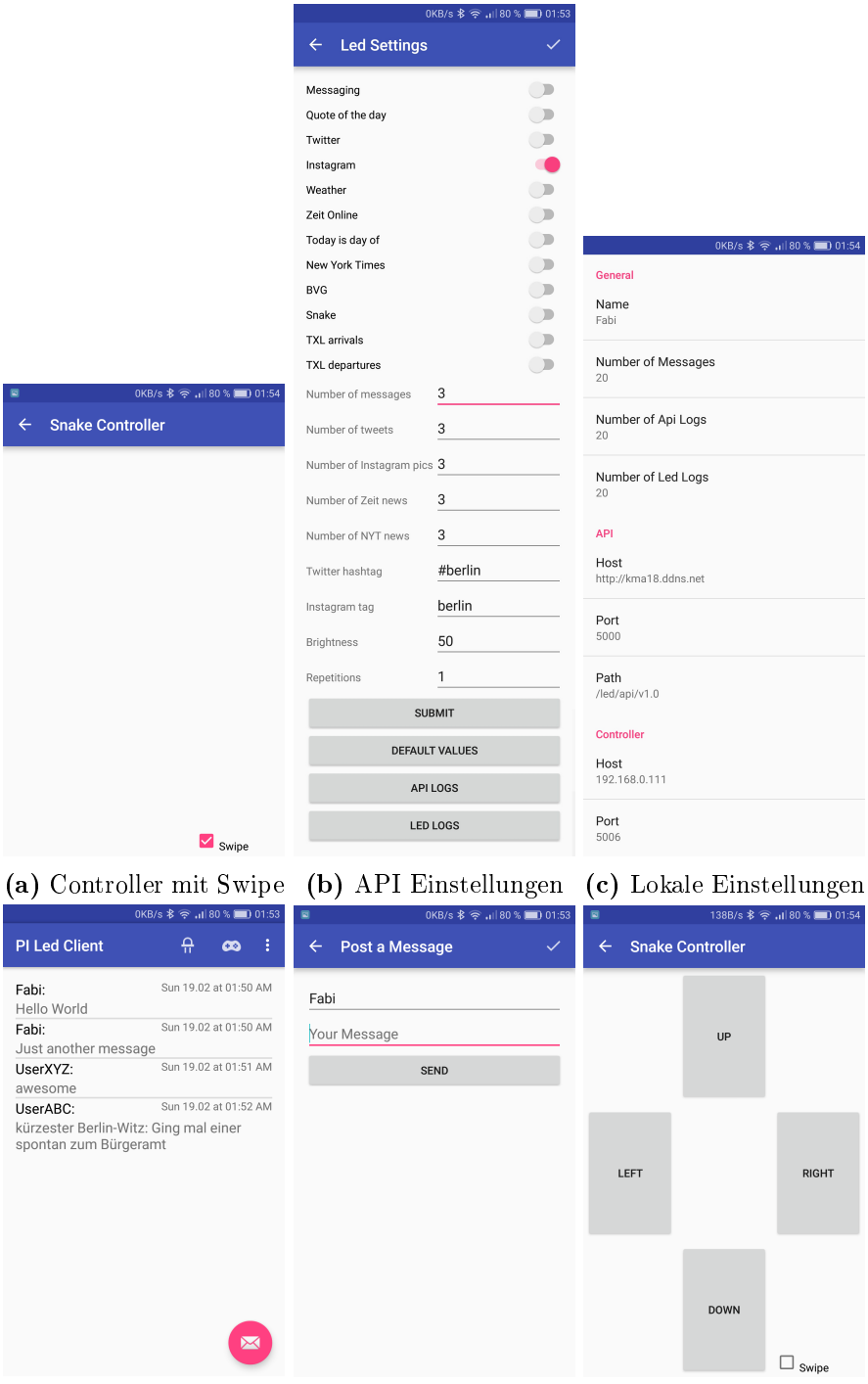
Mitbewohner und Gäste der Wohngemeinschaft sind begeistert. Ich habe während des Indepandant Courseworks viel über Python Programmierung,

der Nutzung von GPIO Pins für IO Aufgaben bis hin zum Web Scrapping gelernt.

Es hat sich jedoch gezeigt, dass der kleine Raspberry Pi 1 Model B mit 700 MHz von der Geschwindigkeit her, schnell an seine Grenzen stößt. Der CPU Benchmark hat gezeigt, dass die CPU mit durchschnittlich 90% ausgelastet ist. Hier wäre sicherlich ein Upgrade auf perfromantere Hardware sinvoll. Desweiteren könnte auch überlegt werden, mehrere solcher LED Panels zu benutzten. Diese können von sich aus in Reihe geschalten werden und das Framework von HZeller kann damit auch umgehen (es sind lediglich ein paar Konfiguration vorzunehmen). Derzeit kann die Helligkeit des Panels via der Smartphone App gesteuert werden. Hier wäre auch ein eigenständiges Hardware Modul (günstig auf dem Markt erhältlich) denkbar, welches die Helligkeit automatisch an der Umgebungshelligkeit anpasst und direkt am Raspberry angeschlossen ist.

10 Impressionen

Es folgt eine Reihe von Bilder um das finale Produkt zu veranschaulichen. Beginnend mit Screenshots aus der Smartphone App, um den Funktionsumfang nochmal aufzuzeigen. Anschließend sind die wie im Abschnitt 7 auf Seite 14 behandelten Untermodule des LED Python Moduls zu sehen. Abschließend ein paar Aufnahmen der Konstruktion in den verschiedenen Ansichten.



(a) Controller mit Swipe **(b)** API Einstellungen **(c)** Lokale Einstellungen

(d) Übersicht aller Nachrichten **(e)** Erstellen einer neuer Nachricht **(f)** Controller mit Buttons

Abb. 7
Screenshots der Smartphone App

Abb. 8
Beispielbilder für die einzelnen Untermodule

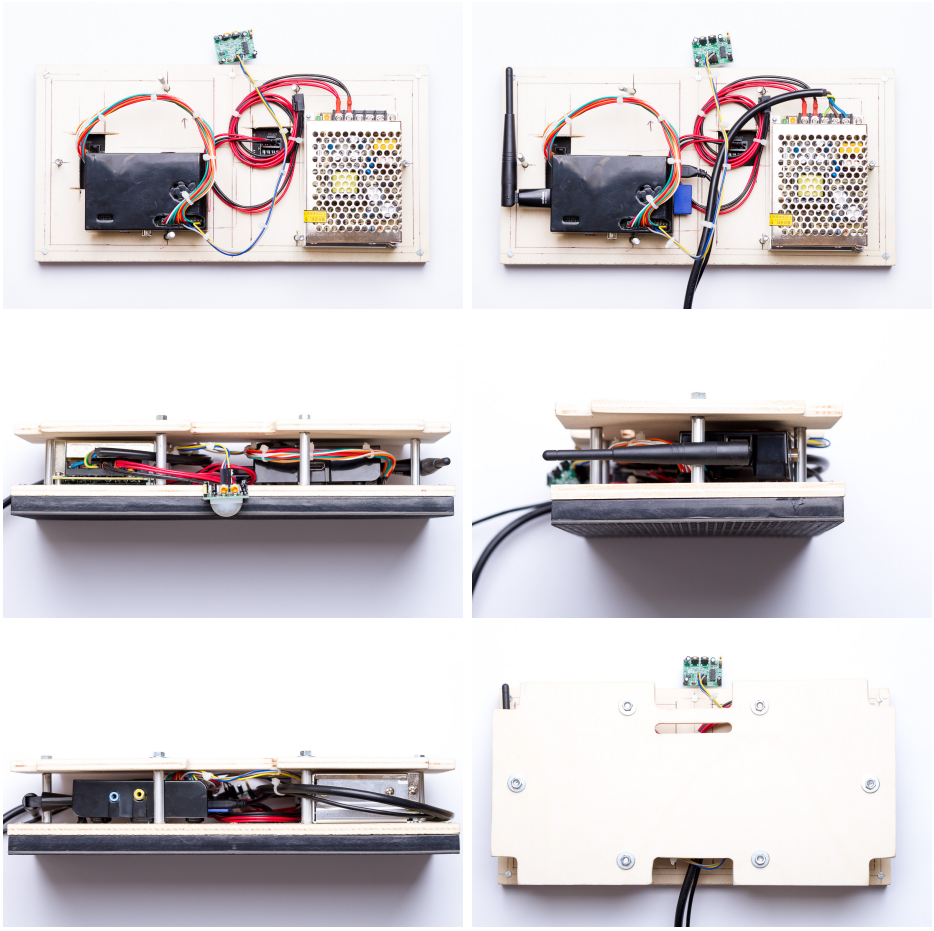


Abb. 9
Ansichten des finalen zusammengesetzten LED Panels

11 Abkürzungsverzeichnis

LED Light Emitting Diode

RGB Farbraum aus Rot, Grün und Blau

DC Gleichstrom

VC Wechselstrom

HUB 75 Interface für LED Displays

MHZ Megahertz

RAM Random Access Memory

WLAN Wireless Local Area Network

GPIO General Purpose Input/Output

USB Universal Serial Bus

IR Infrarot

API Application Programming Interface

UDP User Datagram Protocol

HTTP Hypertext Transfer Protocol

JSON JavaScript Object Notation

TXL Flughafen Tegel

REST Representational State Transfer

FQDN Fully Qualified Domain Name

DNS Domain Name System

IO Input/Output

CPU Central Processing Unit

Literatur

[1] Henner Zeller, „rpi-rgb-led-matrix" Repository auf Github, https://github.com/hzeller/rpi-rgb-led-matrix

[2] The New York Times Developer Network, https://developer.nytimes.com/

[3] They Said So, https://theysaidso.com/

[4] Twitter Developer Documentation, https://dev.twitter.com/rest/public

[5] Twython, https://twython.readthedocs.io/en/latest/

[6] Instagram Developer Documentation, https://www.instagram.com/developer/

[7] OpenWeatherMap, http://openweathermap.org/

[8] Zeit Online, http://www.zeit.de/index

[9] Welcher Tag ist heute, https://welcher-tag-ist-heute.org/

[10] BeautifulSoup, https://www.crummy.com/software/BeautifulSoup/

[11] Verkehrsverbund Berlin-Brandenburg, http://www.vbb.de/de/article/fahrplan/webservices/schnittstellen-fuer-webentwickler/5070.html

[12] Arrivals & Departures TXL, http://www.berlin-airport.de/en/travellers-txl/arrivals-and-departures/departures/index.php

[13] GPIO Python Libary, https://pypi.python.org/pypi/RPi.GPIO

[14] Flask, http://flask.pocoo.org/

[15] Werkzeug - The Python WSGI Utility Library, http://werkzeug.pocoo.org/

[16] Free Dynamic DNS and Managed DNS Provider, http://www.noip.com/