Hochschule Augsburg

Studienfach Hardwaresysteme

»Eulenfunk«

Ein Free Software Internetradio auf Basis des Raspberry Pi

Studenten:

Susanne KIESSLING

Christopher PAHL

Christoph PIECHULA

Dozenten:

Prof. Dr. Hubert HÖGL

M.Sc. Michael SCHÄFERLING

29. Juni 2016

Danksagung: Wir danken Herrn Schäferling von der Hochschule Augsburg für die bereitgestellte Hardware und Unterstützung. Ebenfalls danken wir Herrn Frank Müller und Roland Leuschner von www.rf-elektronik.de für die kompetente Beratung und bereitgestellten Bauelemente. Auch der Firma `ira-kunststoffe` gebührt Dank für die kostenlose Plexiglasplatte.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Vorwort

1.1 Hinweis zu den Vorkenntnissen

Das vorliegende Projekt ist im Rahmen einer Studienarbeit im Fach Hardwaresysteme an der Hochschule Augsburg entstanden. Da die Autoren nicht aus dem Bereich der *Technischen Informatik* sind, wurden jegliche hardwareseitigen Arbeiten nach bestem Grundlagenwissen umgesetzt.

1.2 Namensgebung

Der Name des Projektes ist »`Eulenfunk`«. Passend erschien uns dieser Name, weil er mit dem Spitznamen einer der Autoren zu tun hat. Auch sind unserer Meinung nach Eulen sehr faszinierende Tiere, die durch eine gewisse optische Ästhetik und interessante, manchmal verwirrende, Verhaltensstrukturen auffallen. Der Suffix »-funk« verleiht dem Name zudem durchaus etwas majestätisches.

1.3 Zielsetzung

Das grundlegende Projektziel ist aus vorhandenen alten Hardware–Komponenten ein möglichst optisch und klanglich ansprechendes Internetradio zu entwickeln. Dabei liegt der Schwerpunkt vor allem auch auf einem guten Kosten/Nutzen–Verhältnis. Als Basis für das Projekt dient ein defektes Analog–Radio und ein *Raspberry Pi* aus dem Jahr 2012.

Diese Studienarbeit soll einen Überblick über die verwendeten, beziehungsweise benötigten Komponenten für den Bau eines *Raspberry Pi*–Internetradios verschaffen. Anschließend soll das Wissen für die Ansteuerung bestimmter Hardware–Komponenten mittels der *Raspberry Pi*–GPIO[1] Schnittstelle vermittelt werden.

[1] General-purpose input/output Schnittstelle: https://en.wikipedia.org/wiki/General-purpose_input/output

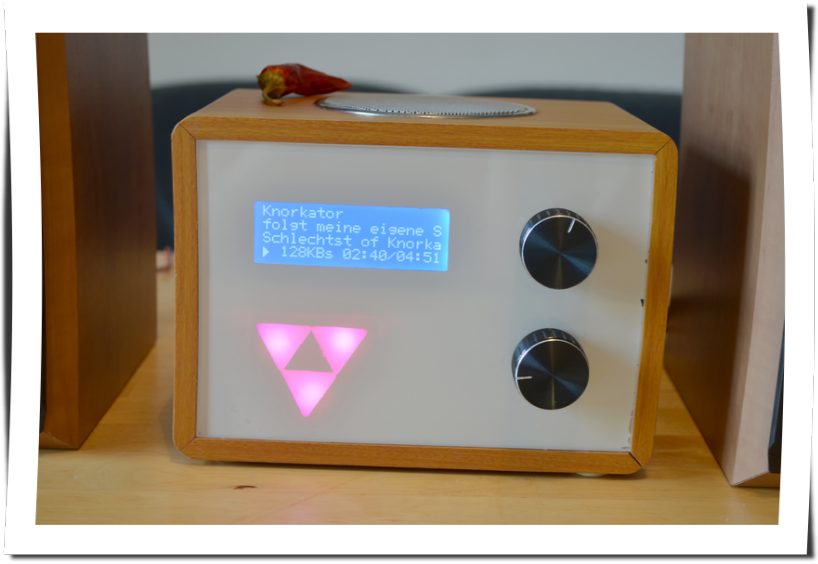

Abbildung 1.1: Foto vom aktuellen Prototypen.

Abbildung 1.1 zeigt den *Eulenfunk* Prototypen, welcher im Zeitraum von drei Wochen im Rahmen des Hardwaresysteme Kür–Projekts entstanden ist. Auf Vimeo[2] ist ein Video des aktuellen Prototypen zu sehen.

1.4 Verwendete Software

Für die Entwicklung und Dokumentation wurden folgende *GNU/Linux* Tools verwendet:

- *Pandoc/LaTeX* (Dokumentation)
- *Vim* (Softwareentwicklung und Dokumentation)
- *Fritzing* (Schaltpläne).
- *Imagemagick* (Bildbearbeitung)

[2] Eulenfunk Prototyp: https://vimeo.com/171646691

2 Motivation

2.1 Private Situation

Die Autoren dieses Projekts leben in einer Wohngemeinschaft zusammen. Die Küche ist der Ort an welchem gemeinsam gekocht, gespeist und diskutiert wird. Für eine angenehme Atmosphäre und als Nachrichten–Quelle sorgte in der Küche früher ein Analog–Radio der Firma *AEG*, welches aufgrund der schlechten Empfangsqualität durch eine Kombination aus »alter Stereoanlage«, »altem Raspberry Pi« und einem »alten Thinkpad X61t« ersetzt wurde. In dieser Kombination fungierte die Stereoanlage als Soundausgabe–Komponente, auf dem *Raspberry Pi* lief der Linux–basierte Player Volumio[1], welcher mit dem Touchscreen des *Thinkpad x61t* über eine Weboberfläche gesteuert wurde. Diese Kombination hat zwar funktioniert, jedoch war sie alles andere als benutzerfreundlich, da zuerst die Stereoanlage und der Laptop eingeschaltet werden mussten und eine WLAN–Verbindung zum *Raspberry Pi*–Player hergestellt werden musste. Diese Situation weckte den Wunsch nach einer komfortableren Lösung, beispielsweise ein Internetradio auf Basis des *Raspberry Pi*.

2.2 Kommerzielle Produkte

Kommerzielle Anbieter von Internetradios gibt es wie Sand am Meer. Die Preisspanne liegt hier zwischen 30 € und mehreren hundert Euro. Der Funktionsumfang sowie die Wiedergabequalität ist hier von Hersteller zu Hersteller und zwischen den verschiedenen Preisklassen sehr unterschiedlich. Einen aktuellen Überblick aus dem Jahr 2016 über getestete Modelle gibt es beispielsweise online unter *bestendrei.de*[2].

Das Problem bei den kommerziellen Anbietern ist, dass man hier jeweils an die vorgegebenen Funktionalitäten des Herstellers gebunden ist. Bei einem Do–It–Yourself–Projekt auf Basis von Freier Software beziehungsweise eines freien Hardwaredesigns, hat man die Möglichkeit alle gewünschten Funktionalitäten — auch Features die von keinem kommerziellen Anbieter unterstützt werden — zu integrieren. Beispiele für Funktionalitäten, welche bei kommerziellen Produkten nur schwer beziehungsweise vereinzelt zu finden sind:

- Unterstützung bestimmter WLAN–Authentifizierungsstandards
- Einhängen von benutzerdefinierten Dateifreigaben wie *Samba*, *NFS*, *SSHFS*
- Unterstützung verschiedener *lossy* und *lossless* Formate *OGG VORBIS*, *FLAC*, u.a.
- Integration verschiedener Dienste wie beispielsweise *Spotify*
- Benutzerdefinierte Anzeigemöglichkeiten (Uhrzeit, Wetter, et cetera.)

[1] Volumio: https://volumio.org/
[2] Test von Internetradios: http://www.bestendrei.de/elektronik/internetradio/

3 Hardware

3.1 Anforderungen

Das Radio soll dem Benutzer folgende Hardwarekonfigurationsmöglichkeiten bieten:

- Anschluss passive Lautsprecher/Kopfhörer möglich
- Lautstärkeregelung über Hardware möglich
- Verwendung des internen Lautsprechers des alten Radios
- Statusinformationen zum aktuellen Lied beispielsweise über ein LCD
- LEDs als Statusanzeige und/oder als Visualisierungsvariante von Musik (vgl. [1])
- USB–Anschlussmöglichkeit für externe Datenträger

3.2 Komponenten und Bauteile

Abbildung 3.1 zeigt eine konzeptuelle Übersichts des Zusammenspiels der einzelnen Komponenten.

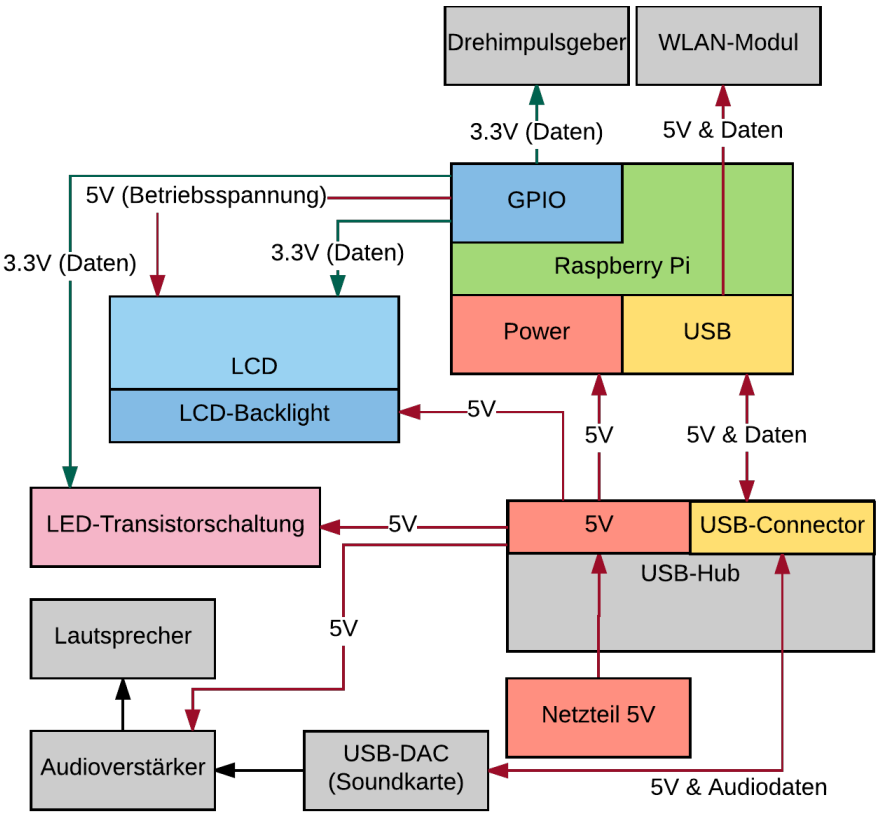

Abbildung 3.1: Grobe Übersicht der verwendeten Komponenten im Zusammenspiel

Folgende Hardwarekomponenten oder Bauteile waren bereits vorhanden oder mussten noch erworben werden:

Vorhanden:

- Altes Gehäuse AEG 4104 Küchenradio[1]
- *Raspberry Pi* aus dem Jahr 2012
- LCD–Anzeige (Altbestände u. Arduino–Kit)
- Kleinbauteile wie LEDs, Widerstände
- USB–Hub für Anschluss von beispielsweise ext. Festplatte
- USB–Soundkarte
- Wi–Fi–Adapter
- Netzteil (diverse 5V, 2A)

Mussten noch erworben werden:

- Audioverstärker (6 €)
- Drehimpulsregler (3 €)
- Kunststoffabdeckung für Front (0 €)
- Farbe (Lack) (5 €)
- Drehknöpfe für das Gehäuse (2 €)
- Schrumpfschläuche (1 €)
- Kippschalter (6 €)

Insgesamt wurden für das Projekt ca. 23 € ausgegeben.

3.3 Raspberry Pi

Der vorhandene *Raspberry Pi* ist aus dem Jahr 2012. Die genaue CPU– und Board–Revision kann auf Linux unter `proc` ausgelesen werden. Von der Hardware–Revision kann auf die *Raspberry Pi*–Revision geschlossen werden, siehe hierzu auch [2], Seite 46:

```
$ cat /proc/cpuinfo
processor       : 0
model name      : ARMv6-compatible processor rev 7 (v6l)
BogoMIPS        : 697.95
Features        : half thumb fastmult vfp edsp java tls
CPU implementer : 0x41
CPU architecture: 7
CPU variant     : 0x0
CPU part        : 0xb76
CPU revision    : 7
```

[1] AEG Küchenradio 4104: https://www.amazon.de/AEG-MR-4104-Desgin-Uhrenradio-buche/dp/B000HD19W8

```
Hardware        : BCM2708
Revision        : 0003
Serial          : 00000000b8b9a4c2
```

Laut Tabelle unter [2], Seite 45 handelt es sich hierbei um das Modell B Revision 1+ mit 256MB RAM.

Je nach Raspberry Revision sind die Pins teilweise unterschiedlich belegt. Seit Modell B, Revision 2.0 ist noch zusätzlich der P5 Header dazu gekommen. Abbildung 3.2[2] zeigt die GPIO–Header des *Raspberry Pi* Modell B Revision 1+.

3.3.1 GPIO–Schnittstelle

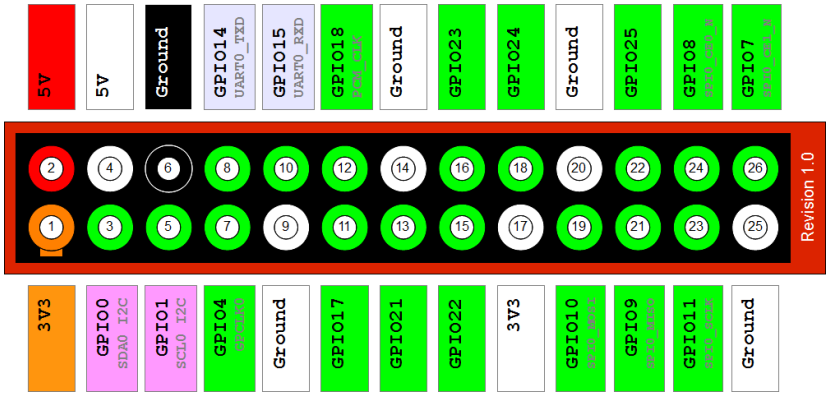

Abbildung 3.2: GPIO-Header des Raspberry Pi Modell B Rev 1.0+

3.3.1.1 GPIO–Pinbelegung und Funktionalität

Die GPIO–Pins des *Raspberry Pi* haben eine Logikspannung von 3.3V und sind pro GPIO–Pin mit max. 16mA belastbar. Der gesamte GPIO–Header sollte mit nicht mehr als 50mA belastet werden, da es darüber hinaus zu Hardwareschäden kommen kann (vgl. [2], Seite 121 ff.).

Die *Logik–Pegel* der GPIO–Pins sind beim *Raspberry Pi* wie folgt definiert [2], Seite 129 ff.:

- $\leq$ 0,8V, input low
- $\geq$ 1,3V, input high

Die Ansteuerung von LEDs über GPIO erfolgt binär. Das heißt, dass die LED entweder aus (low) oder an (high) sein kann.

In der »analogen« Welt ist es jedoch möglich eine LED über das Senken der Spannung zu dimmen. Um ein Dimmen in der digitalen Welt zu erreichen wird ein Modulationsverfahren angewandt, welches Pulsweitenmodulation (*PWM*) heißt. Unter [3], Seite 121 ff. und [4], Seite 421 ff. finden sich weitere Informationen. Software PWM unter [5], Seite 183 ff. zeigt beispielsweise eine 6% CPU–Last pro GPIO–Pin bei einer PWM–Softwareimplementierung. Wie PWM in *Eulenfunk* eingesetzt wird im Software–Kapitel unter dem Punkt 4.5 beleuchtet.

[2] Bildquelle: http://www.raspberrypi-spy.co.uk/2012/06/simple-guide-to-the-rpi-gpio-header-and-pins/#prettyPhoto

3.4 LCD–Anzeige

Um dem Benutzer — beispielsweise Informationen über das aktuell gespielte Lied — anzeigen zu können, soll eine LCD–Anzeige verbaut werden. In den privaten Altbeständen finden sich folgende drei Hitachi–hd44780–kompatible Modelle:

- Blaues Display, 4x20 Zeichen, Bolymin BC2004A
- Blaues Display, 2x16 Zeichen, Bolymin BC1602A
- Grünes Display, 4x20 Zeichen, Dispalytech 204B

Für *Eulenfunk* wurde das blaue 4x20 Display — aufgrund der größeren Anzeigefläche und Farbe — gewählt.

3.4.1 Anschlussmöglichkeiten

Eine LCD–Anzeige kann an den *Raspberry Pi* auf verschiedene Art und Weise angeschlossen werden. Anschlussmöglichkeiten für eine LCD–Anzeige wären beispielsweise:

- GPIO direkt (parallel)
- I2C–Bus (seriell)
- SPI–Bus (seriell)

Die serielle Anschlussmöglichkeit bietet den Vorteil, dass weniger Datenleitungen (GPIO–Pins) verwendet werden. Für den parallelen Betrieb des Displays werden mindestens sechs GPIO–Pins benötigt, für den seriellen Anschluss über I2C lediglich nur zwei.

Da für den seriellen Betrieb beispielsweise über den I2C–Bus zusätzliche Hardware benötigt wird, wird die parallele Ansteuerung über die GPIO–Pins bevorzugt. Weitere Informationen zum seriellen Betrieb über I2C sind unter [6], Seite 61, ff. zu finden.

Das Display arbeitet mit einer Logik–Spannung von 3.3V – 5V. Da die GPIO–Pins jedoch eine High–Logik von 3,3V aufweisen, würde man hier in der Regel einen Pegelwandler bei bidirektionaler Kommunikation und 5V benötigen. Da aber auf das Display nur zugegriffen und die GPIO–Pins nicht schreibend benutzt werden, kann ein Betrieb des Displays auch mit 5V erfolgen. Beim 3.3V Betrieb welcher laut Datenblatt[3] auch möglich sein soll, hatte das Display leider nur eine sehr schwachen beziehungsweise unzureichenden Darstellungskontrast, weswegen der 5V Betrieb gewählt wurde. Zudem wurde an *Pin3* (LCD) ein 100Ω Potentiometer hinzugefügt. Dies ermöglicht den Kontrast variabel einzustellen.

Die Hintergrundbeleuchtung des Displays wurde direkt über ein Potentiometer mit 2KΩ an die 5V Spannungsversorgung angeschlossen. Es wurde hier die direkte Speisung vom Netzteil gewählt, um den GPIO–Header nicht unnötig zu belasten.

Laut Datenblatt kann die Hintergrundbeleuchtung entweder mit 3.4V ohne Vorwiderstand oder mit 5V bei einem 27Ω Widerstand betrieben werden. Damit das Display beim herunter geregeltem

[3] Datenblatt Bolymin BC2004A: http://www.dema.net/pdf/bolymin/BC2004A-series_VER04.pdf

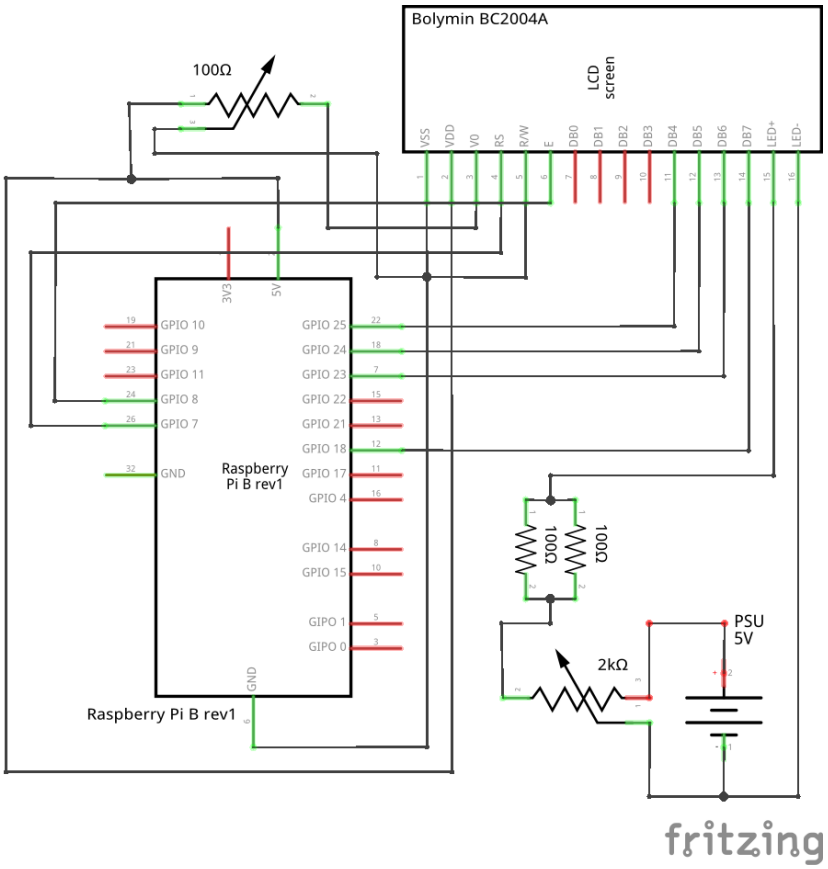

Abbildung 3.3: Verdrahtung von LCD im 4–Bit Modus und Raspberry Pi, alle hierzu benötigten Informationen sind im Datenblatt zu finden.

Potentiometer keinen Schaden nimmt, wurden zusätzlich zwei Widerstände mit 100Ω (parallel geschaltet = 50Ω) zwischen Display und Potentiometer gehängt. Abbildung 3.3 zeigt die Verdrahtung des LCD mit dem *Raspberry Pi*.

Der resultierende Gesamtwiderstand ohne Potentiometer beträgt in diesem Fall $\approx 50\ \Omega$:

$$R_{ges} = \frac{R_1 \times R_2}{R_1 + R_2} = \frac{100\Omega \times 100\Omega}{100\Omega + 100\Omega} = 50\Omega$$

3.5 Drehimpulsgeber

Um eine minimale Anzahl an Bedienelementen zu gewährleisten, wird bei *Eulenfunk* ein Drehimpulsgeber mit Schalter gewählt. Für erste Testzwecke wurde von der Hochschule ein *ALPS STEC12E08* bereitgestellt. Dieser wurde im Laufe der Entwicklung durch einen *ALPS STEC11B09*[4] ersetzt, da dieser mittels Mutter und Schraube am Gehäuse stabiler befestigt werden kann.

Der verwendete Drehimpulsgeber hat insgesamt fünf Anschlüsse. Zwei Signalleitungen (A und B),

[4]Drehimpulsgeber ALPS STEC11B09: https://www.reichelt.de/Drehimpulsgeber/STEC11B09/3/index.html?ACTION=3&GROUPID=3714&ARTICLE=73915

zwei mal *GND* (jeweils für Drehgeber und Schalter) und einen Anschluss für den Schalter. Beim Drehen eines Drehimpulsgebers wird ein Rechtecksignal generiert. Je nach Muster der beiden Datensignale A oder B, kann entschieden werden ob es sich um eine Rechts– oder Linksdrehung handelt. Siehe [7], Seite 361 ff. für weitere Hintergrundinformationen zum Drehimpulsgeber.

Abbildung 3.4 zeigt den Anschluss des Drehimpulsgebers am *Raspberry Pi*.

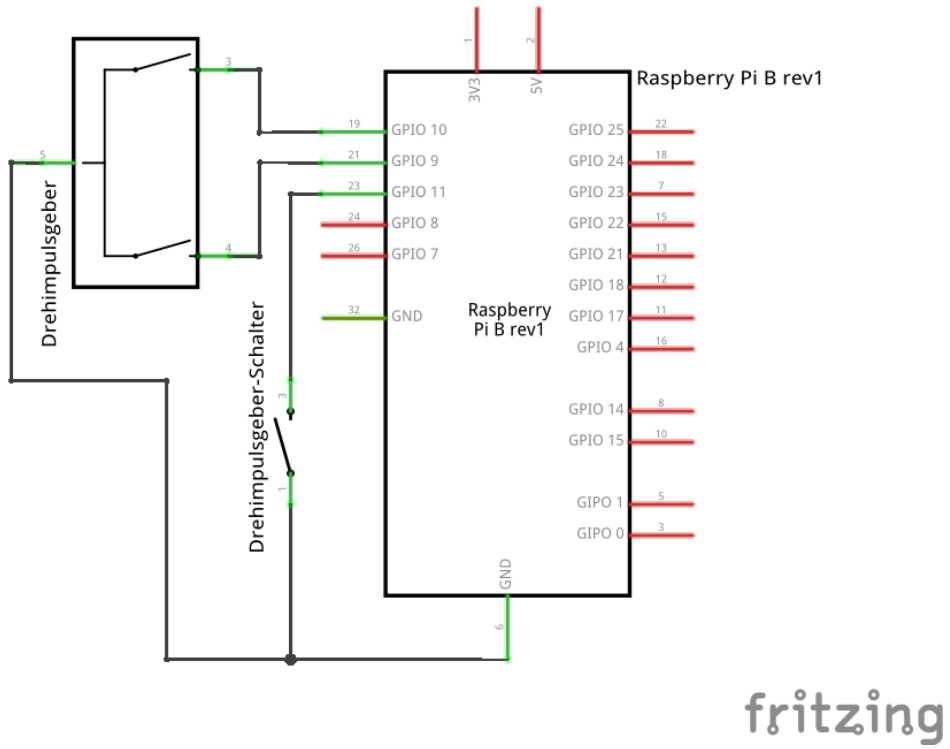

Abbildung 3.4: Drehimpulsgeber–Anschluss an den Raspberry Pi, Abbildung zeigt Kombination aus Potentiometer und Schalter.

3.6 Soundkarte

Die interne Soundkarte des *Raspberry Pi* ist über eine simple Pulsweitenmodulation realisiert. Die einfache Schaltung soll hier laut Internetquellen[5] eine sehr niedrige Audioqualität bieten.

Aus diesem Grund wird bei *Eulenfunk* auf das USB–Audio–Interface *BEHRINGER U-PHONO UFO202*[6] (USB–Soundkarte) gesetzt.

3.7 Audioverstärkermodul

Da eine Soundkarte in der Regel zu wenig Leistung hat, um einem Lautsprecher »vernünftig« anzusteuern, wird ein Audioverstärker benötigt. Da neben dem Anschluss von externen Lautsprechern auch eine Lautstärkeregelung über ein Potentiometer erfolgen soll, ist die Entscheidung einfachheitshalber auf ein Audioverstärker–Modul auf Basis vom PAM8403[7] Stereo-Verstärker mit Potentiometer gefallen. Eine Do–It–Yourself–Alternative wäre ein Transistor–basierter Audio–Verstärker, hier gibt es online diverse Bauanleitungen[8].

Das Audioverstärker–Modul hat folgende Anschlusspins:

- Left–In, Right–In, GND
- 5V+ und GND (Betriebsspannung)
- Left–Side–Out (+), Left–Side–Out (-)
- Right–Side–Out (+), Right–Side–Out (-)

Laut diverser Onlinequellen[9], dürfen die Ausgänge für einen Mono–Betrieb eines auf dem PAM8403–basierten Verstärkers nicht parallel geschaltet werden. Aus diesem Grund kommt ein 4–poliger *EIN-EIN-Kippschalter*[10] zum Einsatz. So kann zwischen dem internen Lautsprecher (Mono–Betrieb) und den externen Stereo Lautsprecher–Anschlüssen sauber per Hardware hin und her geschaltet werden.

Damit im Mono–Betrieb nicht nur ein Kanal verwendet wird, ermöglicht *Eulenfunk* das Umschalten zwischen Mono– und Stereo–Betrieb in Software.

[5] Raspberry Pi onboard Sound: http://www.crazy-audio.com/2013/11/quality-of-the-raspberry-pi-onboard-sound/
[6] BEHRINGER U-PHONO UFO202 Audio Interface: http://www.produktinfo.conrad.com/datenblaetter/1300000-1399999/001370864-an-01-de-BEHRINGER_UFO_202_AUDIOINTERFACE.pdf
[7] Verstärkermodul: https://www.amazon.de/5V-Audioverstärker-Digitalendstufenmodul-Zweikanalige-Stereo-Verstärker-Potentiometer/dp/B01ELT81A6
[8] Transistor–Verstärker: http://www.newsdownload.co.uk/pages/RPiTransistorAudioAmp.html
[9] PAM8403 Mono–Betrieb: http://electronics.stackexchange.com/questions/95743/can-you-bridge-or-parallel-the-outputs-of-the-pam8403-
[10] Kippschalter 4–polig EIN–EIN: http://www.reichelt.de/Kippschalter/MS-500P/3/index.html?&ACTION=3&LA=2&ARTICLE=13172&GROUPID=3275&artnr=MS+500P

3.8 LED–Transistorschaltung

Die Ansteuerung einer LED mittels GPIO–Pin ist recht simpel. Sollen jedoch mehrere LEDs angesteuert werden, so wird in der Regel pro LED ein GPIO–Pin benötigt. LEDs sollten nie ohne Vorwiderstand an den *Raspberry Pi* angeschlossen werden, da durch den hohen Stromfluss die LED beschädigt werden könnte. Weiterhin muss bei LEDs auch auf die Polung geachtet werden, die abgeflachte Seite — meist mit dem kürzerem Beinchen — ist in der Regel die Kathode (Minuspol). Abbildung 3.5 zeigt exemplarisch den Anschluss einer *classic LED rot*[11], mit einer Flussspannung von $U_{LED} \approx 2V$, die mit einem Strom von I_{LED} = 20 mA gespeist werden soll. Die Berechnung des Vorwiderstandes erfolgt nach folgender Formel:

$$R_{LED} = \frac{U_{GPIO} - U_{LED}}{I_{LED}} = \frac{3.3V - 2V}{20mA} \approx 65\Omega$$

Hinweis: Da ein GPIO–Pin aber mit nur max. 16mA belastet werden sollte, sollte in unserem Beispiel durch 16mA anstatt 20mA geteilt werden um den max. Stromfluss auf 16mA zu begrenzen. In diesem Fall würden wir auf $\approx 82\Omega$ kommen.

Da Widerstände meistens in fest vorgegebenen Größen vorhanden sind, kann im Fall eines nicht exakt existierenden Widerstandswertes einfach der nächsthöhere Widerstandswert genommen werden. Im Beispiel wird ein 100Ω Widerstand verwendet.

Weitere Beispiele und Grundlagen zur Reihen– und Parallelschaltung von LEDs können online beispielsweise unter *led-treiber.de*[12] eingesehen werden.

Je nach Typ und Farbe ist der benötigte Strom um ein vielfaches höher wie in unserem Beispiel. Die in 3.5 abgebildete LED kann vom GPIO–Pin nur einen max. Strom von 16 mA beziehen

In *Eulenfunk* sollen mehrere intensiv leuchtende LEDs verbaut werden. Da die GPIO–Pins in ihrer Leistung sehr begrenzt sind, würde es sich anbieten eine externe Stromquelle zu verwenden. Um die Speisung über eine externe Stromquelle zu ermöglichen, kann eine Transistorschaltung verwendet werden (vgl. [8], Seite 219 ff.).

Für die Transistorschaltung wurden von Seite der Hochschule Augsburg NPN– (BC547C) und PNP–Transistoren (BC557C) bereitgestellt. Für den ersten Testaufbau wurde der PNP–Transistor und eine RGB–LED[13] mit gemeinsamen Minuspol verwendet. Dabei ist aufgefallen, dass die LED ständig geleuchtet hat. Eine kurze Recherche hat ergeben, dass der Transistor permanent durchgeschaltet war, weil die Spannung an der Basis (GPIO–Pin, 3,3V) geringer war als die Betriebsspannung für die LED (5V).

Der zweite Testaufbau mit dem NPN–Transistor BC547C und einer RGB–LED[14] mit gemeinsamen Pluspol hat das gewünschte Ergebnis geliefert.

[11] Datenblatt mit verschiedenen LED–Typen: https://www.led-tech.de/de/5mm-LEDs_DB-4.pdf
[12] Beispiele zur Ansteuerung von LEDs: http://www.led-treiber.de/html/vorwiderstand.html
[13] RGB-LED Common Cathode: http://download.impolux.de/datasheet/LEDs/LED0870RGB5mmklar10000mcd.pdf
[14] RGB-LED Common Anode: http://download.impolux.de/datasheet/LEDs/LED09258RGB5mmklar10000mcd_GP.pdf

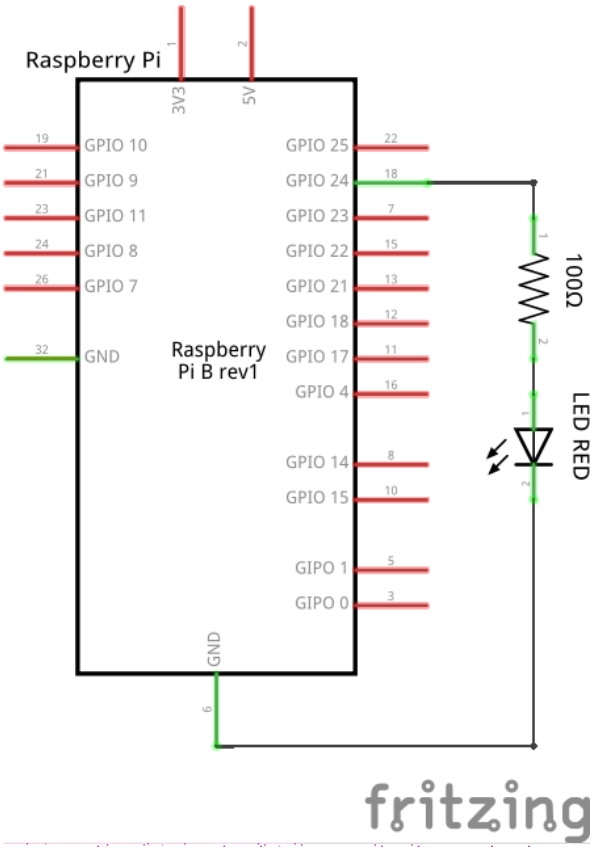

Abbildung 3.5: Anschluss einer roten LED mit Vorwiderstand am Raspberry Pi GPIO–Pin

Da der Hersteller für die von der Hochschule bereitgestellten Transistoren unbekannt ist, wurden typische Durchschnittswerte für die Dimensionierung der restlichen Bauteile verwendet.

Wie es aussieht sind die meisten BC547C Transistor–Typen für einen max. Strom I_{CE}=100 mA konstruiert. Für die Berechnung des Basis–Vorwiderstandes wird der Stromverstärkungsfaktor h_{FE}[15] benötigt. Je nach Hersteller variieren die Werte zwischen 200[16] und 400[17]. Da der maximale Laststrom I_{CE} pro Transistor 60 mA (3 LEDs je max. 20mA) beträgt, sieht die Berechnung des Basisstroms — bei einem durchschnittlichem h_{FE} = 300 — wie folgt aus:

$$I_{Basis} = \frac{I_{CE}}{h_{FE}} = \frac{0.06A}{300} \approx 200\mu A$$

Der BC547C Transistor benötigt eine durchschnittliche U_{BE} = 0,7V zum Durchschalten. Die GPIO-Pins des *Raspberry Pi* haben einen Spannungspegel von 3.3V. Daraus ergibt sich folgende Berechnung des Basis–Vorwiderstandes:

$$R_{Basis} = \frac{U_{GPIO} - U_{Basis}}{I_{Basis}} = \frac{3,3V - 0,7V}{200\mu A} = 13k\Omega$$

[15] Stromverstärkungsfaktor: http://www.learningaboutelectronics.com/Articles/What-is-hfe-of-a-transistor
[16] Farichild Semiconductor: https://www.fairchildsemi.com/datasheets/BC/BC547.pdf
[17] SEMTECH: http://pdf1.alldatasheet.com/datasheet-pdf/view/42386/SEMTECH/BC547.html

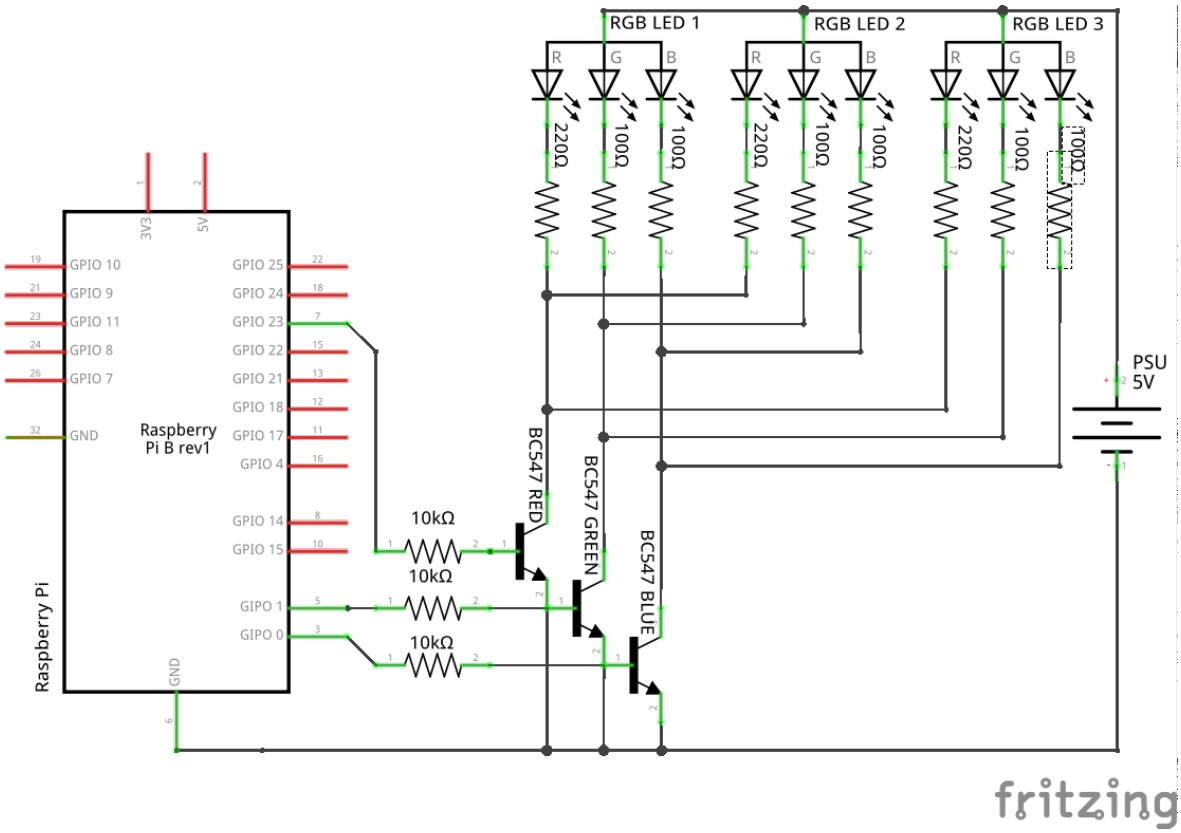

Abbildung 3.6: Transistor–RGB–LED Schaltung

Damit der Transistor jedoch *sicher* durchschaltet, werden Wiederstände mit $10k\Omega$ verwendet. Die in Abbildung 3.6 gelisteten LED–Vorwiderstände ergeben sich aufgrund der verschiedenen Spannungen der unterschiedlichen Farben[18]. Die Berechnung für den Vorwiderstand pro LED schaut am Beispiel der Farbe blau ($U_{LED} = 3,15V, I_{LED} = 20mA$) wie folgt aus:

$$R_{LED} = \frac{U_{Betriebsspannung} - U_{LED}}{I_{LED}} = \frac{5V - 3,15V}{20mA} = 92.5\Omega \approx 100\Omega$$

Analog errechnet sich für die Farbe rot ein Vorwiderstand von 145Ω und für grün ein Vorwiderstand von 87Ω.

3.9 USB–Hub und Netzteil

Der *Raspberry Pi* hat in unserer Revision nur zwei USB–Schnittstellen, diese sind bereits durch die Hardware–Komponenten USB–DAC (Soundkarte) und das Wi–Fi–Modul belegt. Um den Anschluss eines externen Datenträgers, auch mit größerer Last wie beispielsweise einer Festplatte zu ermöglichen, wird ein aktiver USB–Hub benötigt.

[18] RGB-LED Common Anode: http://download.impolux.de/datasheet/LEDs/LED09258RGB5mmklar10000mcd_GP.pdf

Für diesen Einsatzzweck wird aus den Altbeständen ein *LogiLink 4 Port USB 2.0 HUB*[19] verwendet. Viele billige Hubs arbeiten hier entgegen der USB–Spezifikation und speisen den *Raspberry Pi* zusätzlich über die USB–Schnittstelle. Dieses Verhalten wurde bemerkt, als der *Raspberry Pi* ohne Power–Connector alleine nur mit der USB–Verbindung zum USB–Hub bootete.

Bei der Speisung über die USB–Schnittstelle wird die interne Sicherungsschaltung des *Pi* umgangen, deswegen wird in der Regel von einem Betrieb eines USB–Hub mit *backfeed* abgeraten (vgl . [9], Seite 26 ff.). Für den Prototypen wird jedoch der genannte USB–Hub und das dazugehörige Netzteil für den Betrieb von *Eulenfunk* verwendet. Das Netzteil ist für 5V bei max. 2A ausgelegt.

Nachtrag: Die Speisung über das Netzteil des USB–Hubs ist recht instabil. Bei Lastspitzen kommt es anscheinend zu Störeinwirkungen, die sich auf die GPIO–Peripherie auswirken (LCD–Anzeige rendert inkorrekt). Ein weiterer Punkt sind Störfrequenzen, welche teilweise in Form von Störgeräuschen die Audioausgabe überlagern (Hintergrundgeräusche beim Einschalten aller LEDs). Insgesamt wurden drei Netzteile — jeweils 5V, 2A —ausprobiert. Von diesen war lediglich ein einziges als »akzeptabel« einzustufen. Die restlichen zwei führen bei Lastspitzen zu Problemen (Abstürze, fehlerhaftes Rendering auf Display, GPIO–Flips, et cetera). Das *backfeed* des USB–Hubs scheint die genannten Probleme teilweise zu verstärken (vgl . [9], Seite 27).

3.10 Gehäuse

3.10.1 Vorderseite

Abbildung 3.7 zeigt ein Muster der Gehäusefront–Farbe hellelfenbeinweiß RAL 1015. Dieser Farbton wird für die Front verwendet, um *Eulenfunk* einen dezenten »Retro«–Look zu verpassen.

Abbildung 3.7: Muster RAL1015, hellelfenbeinweiß

Das Plexiglas für die Front wurde von der Firma *ira-Kunststoffe* in Schwarzenbach/Saale zugeschnitten. In der Plexiglasfront wurden mit Hilfe von Herrn Schäferling zwei 5mm Löcher (Drehimpulsgeber, Lautstärkeregler–Poti) gebohrt. Anschließend wurde die Plexiglas–Front von der Innenseite lackiert[20], hierbei wurden die Flächen für LCD und die drei LEDs abgeklebt. Zudem werden schwarze Knöpfe in Alu–Optik mit ⌀ 30mm für den Lautstärkeregler und den Drehimpulsgeber verwendet.

[19] LogiLink USB–Hub: https://www.amazon.de/LogiLink-4-Port-Hub-Netzteil-schwarz/dp/B003ECC6O4
[20] Buntlack, hellelfenbein: http://www.obi.de/decom/product/OBI_Buntlack_Spray_Hellelfenbein_hochglaenzend_150_ml/3468725

3.10.2 Rückseite

Für die Rückseite wird die alte Abdeckung des AEG–Radios verwendet. Diese musste teilweise leicht modifiziert werden. An dieser befinden sich zwei Potis für Kontrastregelung und Hintergrundbeleuchtung des LCD, eine USB–Female–Kabelpeitsche, zwei Cinch Stecker für externe Lautsprecher und ein Kippschalter zum Umschalten zwischen internen und externen Lautsprechern.

3.11 Betriebssystem

Mittlerweile gibt es für den *Raspberry Pi* viele offiziell zugeschnittene Betriebssysteme (vgl. [10], Seite 29 ff. und [11], Seite 47 ff.). Bei den Linux–Distributionen ist *Raspbian* eine der bekanntesten Distribution – welche auf *Debian* basiert. *Raspbian* bringt ein komplettes Linux–basiertes System mit grafischer Benutzeroberfläche mit sich.

Neben den unter [10], Seite 29 ff. genannten Distributionen gibt es mittlerweile auch Windows 10 IoT (Internet of Things) für den *Raspberry Pi*. Dieses speziell für den Embedded Bereich ausgerichtete Windows benötigt jedoch eine ARMv7–CPU als Mindestanforderung[21], was den »alten Raspberry« ausschließt. Außerdem wäre für uns eine proprietäre Lösung ein K.O.–Kriterium, da diese alle Vorteile von Freier Software zunichte machen würde.

3.11.1 Wahl des Betriebssystem

Arch Linux ARM[22] ist eine minimalistische und sehr performante Linux–Distribution welche im Gegensatz zu *Raspbian* ohne Desktop–Umgebung geliefert wird (vgl. [12], Seite 13 ff.) Darüber hinaus ist *Arch Linux* ein bekannter Vertreter von Rolling–Release–Distributionen. Ein weiterer Vorteil für unseren Einsatzzweck ist bei *Arch Linux* das *AUR* (Arch User Repository)[23], dieses erlaubt es eigene Software auf eine schnelle und unkomplizierte Weise der Allgemeinheit zur Verfügung zu stellen.

3.11.2 Einrichtung des Grundsystems

Nach der Installation[24] und dem ersten Booten des Grundsystems muss die Netzwerk–Schnittstelle konfiguriert werden. Arch Linux ARM bietet mit *netctl* eine Profil–basierte Konfigurationsmöglichkeit. Ein Profil kann über das *ncurses*–basierte Tool `wifi-menu` erstellt werden. In unserem Fall wurde das Profil `wlan0-Phobos` erstellt. Anschließend kann das erstellte Profil mit *netctl* verwendet werden.

[21] Systemanforderungen:http://raspberrypi.stackexchange.com/questions/39715/can-you-put-windows-10-iot-core-on-raspberry-pi-zero
[22] Arch Linux ARM: https://archlinuxarm.org/
[23] Arch User Repository: https://aur.archlinux.org/
[24] Arch Linux Installation für Raspberry Pi: https://archlinuxarm.org/platforms/armv6/raspberry-pi#installation

Auflistung der bekannten Profile

```
[root@eulenfunk ~]$ netctl list
  eth0-static
  wlan0-Phobos
```

Aktivierung des gewünschten Profils

```
# Starten des gewünschten Profils
[root@eulenfunk ~]$ netctl start wlan0-Phobos
[root@eulenfunk ~]$ netctl list
  eth0-static
* wlan0-Phobos

# Profil über System-Reboot hinweg aktivieren
[root@eulenfunk ~]$ netctl enable wlan0-Phobos
```

Nun verbindet sich der *Raspberry Pi* nach dem Hochfahren jedes Mal automatisch mit dem Profil `wlan0-Phobos`.

3.12 Erweiterungen und alternative Ansätze

3.12.1 Allgemein

Der aktuelle Prototyp hat lediglich nur ein Potentiometer um die Hintergrundbeleuchtung des LCD zu regeln. Ein anderer Ansatz wäre der Einsatz eines Relais, welches es ermöglichen würde die LCD–Hintergrundbeleuchtung Software–seitig ein– und auszuschalten. Die Software könnte dann automatisch nach längerer Inaktivtät die Beleuchtung dimmen.

3.12.2 Audio–Visualisierung

Beim Projekt *Eulenfunk* wird die Visualisierung von Musik aufgrund der begrenzten Zeit und Hardwareressourcen des *Raspberry Pi* über eine vorberechnete Moodbar–Datei realisiert. Dieser Ansatz funktioniert bei nicht live gestreamter Musik gut. Bei live gestreamter Musik könnte für die Visualisierung eine Fast–Fourier–Transformation in Echtzeit durchgeführt werden. Da jedoch die Ressourcen des *Raspberry Pi* sehr begrenzt sind, sollte hier auf die Verwendung einer GPU–beschleunigten–FFT zurückgegriffen werden (vgl. [13], Seite 657 ff.).

Ein alternativer Ansatz wäre auch die Realisierung einer Musik–Visualisierung mittels Hardwarekomponenten. Ein möglicher Ansatz aus Hardware–basierten Hochpass– und Tiefpassfiltern in Form einer Disco–Beleuchtung wird unter [7], Seite 261 ff. beschrieben.

3.12.3 Echtzeituhr

Der *Raspberry Pi* besitzt keine Hardware–Uhr. Aufgrund der Tatsache, dass es sich bei *Eulenfunk* um ein Internetradio handelt wurde auf eine Echtzeituhr (real time clock, RTC) verzichtet, da sich die Uhr von *Eulenfunk* aufgrund der permanenten Internetverbindung mittels NTP[25] über das Internet synchronisieren kann. Eine Erweiterung um eine Echtzeituhr wird in [6], Seite 145 ff. und [5], Seite 77 ff. ausführlich beschrieben. Mit einer RTC wäre die Implementierung einer Weckerfunktion möglich, welche das Radio gezielt zu einer bestimmten Uhrzeit aufweckt und Musik abspielt.

3.12.4 Fernbedienung

Eine weitere Erweiterung wäre die Integration einer Fernbedienung. Diese ließe sich relativ einfach mittels eines Infrarot–Sensors und beispielsweise der *lirc*–Bibliothek umsetzen. Siehe auch [11], Seite 190 ff. für weitere Informationen.

3.12.5 Batteriebetrieb

Da die Strom– beziehungsweise Spannungsversorgung beim *Raspberry Pi* problematisch ist, wäre auch ein Batterie– beziehungsweise Akkubetrieb möglich. Eine einfache Schaltung für einen Batteriebetrieb würde sich beispielsweise mit einem *LM7805*–Spannungsregler oder einem Abwärtswandler realisieren lassen ([4], Seite 24 ff.).

3.13 Fazit

Grundsätzlich kann der Hardware–Prototyp als erfolgreich umgesetzt betrachtet werden. Die geplanten Anforderungen an die Hardware konnten soweit alle umgesetzt werden. Jedoch sollte gesagt werden, dass das Netzteil und der USB–Hub — wie bereits im jeweiligem Kapitel erläutert — aktuell ein eher instabiles Verhalten aufweisen. Hier ist noch eine Umstellung auf einen besseren USB–Hub (und Netzteil mit 5V, 3A) ohne *backfeed* nötig.

Auf der Seite des Betriebssystems wäre die relativ junge Linux–Distribution *Alpine Linux*[26] eine mögliche Verbesserung für den Einsatzzweck Internetradio. Diese Distribution hat ihren Fokus auf Ressourceneffizienz und Systemsicherheit. Ein weiterer Vorteil wäre der `diskless mode`, welcher das komplette Betriebssystem in den Arbeitsspeicher lädt. In diesem Modus müssen Änderungen mit einem *Alpine Local Backup (lbu)*–Tool explizit auf die Festplatte geschrieben werden. Das hätte den Vorteil, dass man die Abnutzung des Flash–Speichers, durch unnötige Schreib/Lese–Vorgänge, minimieren würde. Momentan unterstützt diese Distribution allerdings noch nicht das von uns favorisierte `systemd`.

[25] Network Time Protocol: https://de.wikipedia.org/wiki/Network_Time_Protocol
[26] Alpine Linux für Raspberry Pi: https://wiki.alpinelinux.org/wiki/Raspberry_Pi

4 Software

Dieses Kapitel beleuchtet die einzelnen Software–Komponenten von *Eulenfunk*. Selbst verfasste Komponenten sind dabei als solche gekennzeichnet.

4.1 Anforderungen

Zu Beginn der Entwicklung wurden einige Anforderungen an die Software festgelegt:

1. Leichte Bedienbarkeit, obwohl es nur einen Knopf gibt, der nur die Aktionen »links drehen«, »rechts drehen« und Drücken zulässt.
2. Die Software sollte vernünftig mit den Hardwareressourcen des *Raspberry Pi* umgehen. Dabei sollte die Hardware nicht die ganze Zeit auf volle Last laufen, um eine Überhitzung im beengten Gehäuse zu vermeiden.
3. Die Software sollte möglichst ausfallsicher sein. Fällt beispielsweise ein Feature durch einen Absturz der Software aus, so sollten andere Teile des Radios nach Möglichkeit nicht betroffen sein. Auch sollte der abgestürzte Teil sich neu starten können und entsprechende Log–Nachrichten hinterlassen. Dienste, die von dem abgestürzten Dienst abhängen, sollten sich bei Neustart dessen wieder neu verbinden.
4. Leichte Wartbarkeit und Fehlersuche durch Schreiben von Logs.
5. Einfache Erweiterbarkeit und Integrierbarkeit mit anderen Anwendungen durch lose Kopplung von Diensten.

Inwieweit und wodurch diese Anforderungen erfüllt worden sind, wird im Fazit erläutert.

4.2 Abspielsoftware

Für den Betrieb des Internetradios soll der MPD (Music–Player–Daemon) verwendet werden, da *Eulenfunk* auf einem eigens entwickeltem MPD–Client basieren soll. Andere Projekte greifen oft auf Abspielsoftware wie den *MOC* [10], Seite 189 ff. oder *mplayer* [8] Seite 638 ff. zu.

Der MPD ist ein unter Unix gern genutzter Daemon zum Verwalten und Abspielen von Musik und Radiostreams. Er unterstützt dabei eine große Anzahl von Formaten und kann diese an mehrere Backends wie ALSA, Pulseaudio oder als HTTP–Stream ausgeben (siehe auch Abbildung 4.1). Für unseren Einsatzzweck ist er dabei aus zwei Gründen besonders geeignet: Auch bei sehr großen Sammlungen mit einer 5-stelligen Anzahl von Liedern läuft er problemlos und mit einem vergleichsweise niedrigen Speicherverbrauch. Der zweite Grund ist die lose Kopplung zwischen Abspielsoftware und User–Interface: MPD selbst ist nur ein Daemon, der mittels eines zeilenbasierten Textprotokolls

(dem MPD–Protokoll[1]) steuerbar ist. Um die Bedienoberfläche kümmert sich dann ein separater MPD–Client, welcher als »Fernbedienung« für den Daemon fungiert.

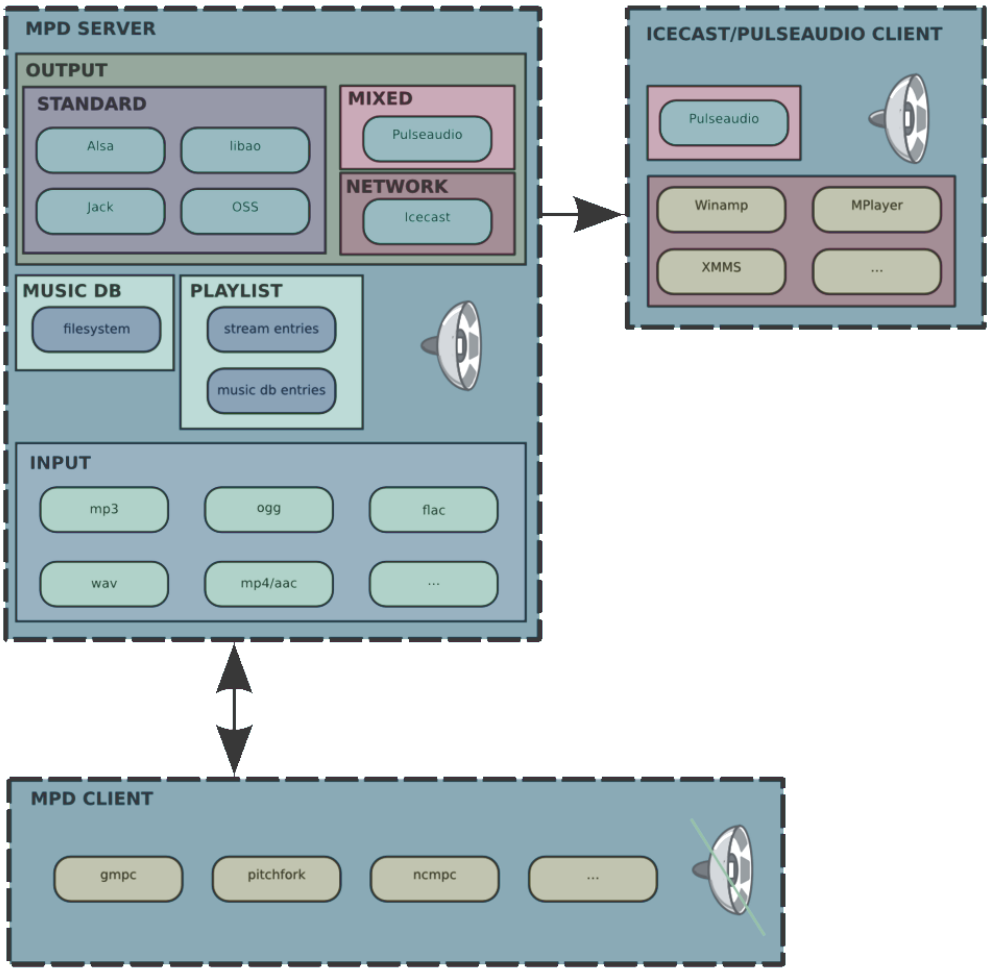

Abbildung 4.1: Übersicht über die Architektur des MPD–Daemons (Quelle: http://mpd.wikia.com/wiki/What_MPD_Is_and_Is_Not)

4.3 Softwarearchitektur

Die »Nachbaubarkeit« vieler Bastelprojekte ist häufig durch die Software recht eingeschränkt, da diese entweder nicht frei verfügbar ist oder zu wenig generisch ist als dass man die Software leicht auf das Projekt anpassen könnte. Meist handelt es sich dabei um ein einziges, großes C–Programm oder ein eher unübersichtliches Python–Skript. Aus diesem Grunde soll die Software für *Eulenfunk* derart modular aufgebaut sein, dass einzelne Module problemlos auch auf andere Projekte übertragbar sind und später eine leichte Erweiterbarkeit gewährleistet ist. Damit auch andere die Software einsetzen können wird sie unter die GPL in der Version 3 (vgl. [14]) gestellt.

Zu diesem Zweck ist die Software in zwei Hauptschichten unterteilt. Die untere Schicht bilden dabei

[1] Details unter https://www.musicpd.org/doc/protocol

die *Treiber*, welche die tatsächliche Ansteuerung der Hardware erledigen. Dabei gibt es für jeden Teil der Hardware einen eigenen Treiber, im Falle von *Eulenfunk* also ein separates Programm für die LCD-Ansteuerung, das Setzen der LED–Farbe und dem Auslesen des Drehimpulsgebers.

Die Schicht darüber bilden insgesamt fünf einzelne Dienste, die über eine Netzwerkschnittstelle angesprochen werden und jeweils eine Funktionalität des Radios umsetzen. So gibt es beispielsweise einen Dienst, der die Ansteuerung des LCD–Displays *komfortabel* macht, ein Dienst, der die LEDs passend zur Musik einfärbt und ein Dienst, der automatisch eine Playlist aus der Musik auf angesteckten externen Speichermedien erstellt. Die jeweiligen Dienste sprechen mit den Treibern indem sie Daten auf `stdin` schreiben, bzw. Daten von `stdout` lesen. Um die Dienste auf neue Projekte zu portieren, ist also nur eine Anpassung oder Erweiterung der jeweiligen Treiber notwendig.

Der Vorteil liegt dabei klar auf der Hand: Die lose Kopplung der einzelnen Dienste erleichtert die Fehlersuche ungemein und macht eine leichte Austauschbarkeit und Übertragbarkeit der Dienste in anderen Projekte möglich. Stellt man beispielsweise fest, dass der Prozessor des Radios voll ausgelastet ist, so kann man mit Tools wie `htop` einfach herausfinden welcher Dienst dafür verantwortlich ist.

4.3.1 Sprachwahl

Die momentane Software ist in den Programmiersprachen *C* und *Go* geschrieben. Dazu kommt lediglich ein einzelnes Bash–Skript zum Auslesen von Systeminformationen.

Die Ressourcen auf dem *Raspberry Pi* sind natürlich sehr limitiert, weswegen sehr speicherhungrige Sprachen wie Java oder Ähnliches von vornherein ausschieden. Obwohl Python nach Meinung des Autors eine schöne und komfortable Sprache ist und viele gute Bibliotheken für den *Pi* bietet, schied es ebenfalls aus diesem Grund aus.

Ursprünglich war sogar geplant, alles in *Go* zu schreiben. Leider gibt es nur wenige Pakete für die GPIO–Ansteuerung und auch keine Bibliothek für softwareseitige Pulsweitenmodulation. Zwar hätte man diese notfalls auch selbst mittels `/sys/class/gpio/*` implementieren können, doch bietet *Go* leider auch keine native Möglichkeit mit Interrupts zu arbeiten. Wie später beschrieben ist dies allerdings für den Treiber nötig, der den Drehimpulsgeber ausliest.

Für *Go* sprechen ansonsten folgende Gründe als Sprache für die höhere Logik:

- **Garbage Collector:** Erleichtert die Entwicklung lang laufender Dienste.
- **Hohe Grundperformanz:** Zwar erreicht diese nicht die Performanz von C, liegt aber zumindest in der selben Größenordnung (vgl. [15], S. 37).
- **Weitläufige Standardbibliothek:** Kaum externe Bibliotheken notwendig.
- **Schneller Kompiliervorgang:** Selbst große Anwendungen werden in wenigen Sekunden in eine statische Binärdatei ohne Abhängigkeiten übersetzt.
- **Kross–Kompilierung:** Durch Setzen der `GOARCH=arm` Umgebungsvariable kann problemlos auf einen x86-64–Entwicklungsrechner eine passende ARM–Binärdatei erzeugt werden.
- **Eingebauter Scheduler:** Parallele und nebenläufige Anwendungen wie Netzwerkserver sind sehr einfach zu entwickeln ohne für jede Aufgabe einen neuen Thread starten zu müssen.

- Ein Kriterium war natürlich auch, dass die Autoren gute Erfahrung mit der Sprache hatten und **neugierig** waren, ob sie auch für solche Bastelprojekte gut einsetzbar ist.

C ist hingegen für die Entwicklung der Treiber vor allem aus diesen Gründen eine gute Wahl:

- Programmierung mit **Interrupts** bequem und nativ möglich.
- Hohe **Performanz** und genaue Kontrolle über den Speicherverbrauch.
- Verfügbarkeit von **wiringPi**.

4.4 Überblick der einzelnen Komponenten

Ein Überblick über die existierenden Dienste liefert Abbildung 4.2. Die einzelnen Komponenten werden im Folgenden detailliert erläutert.

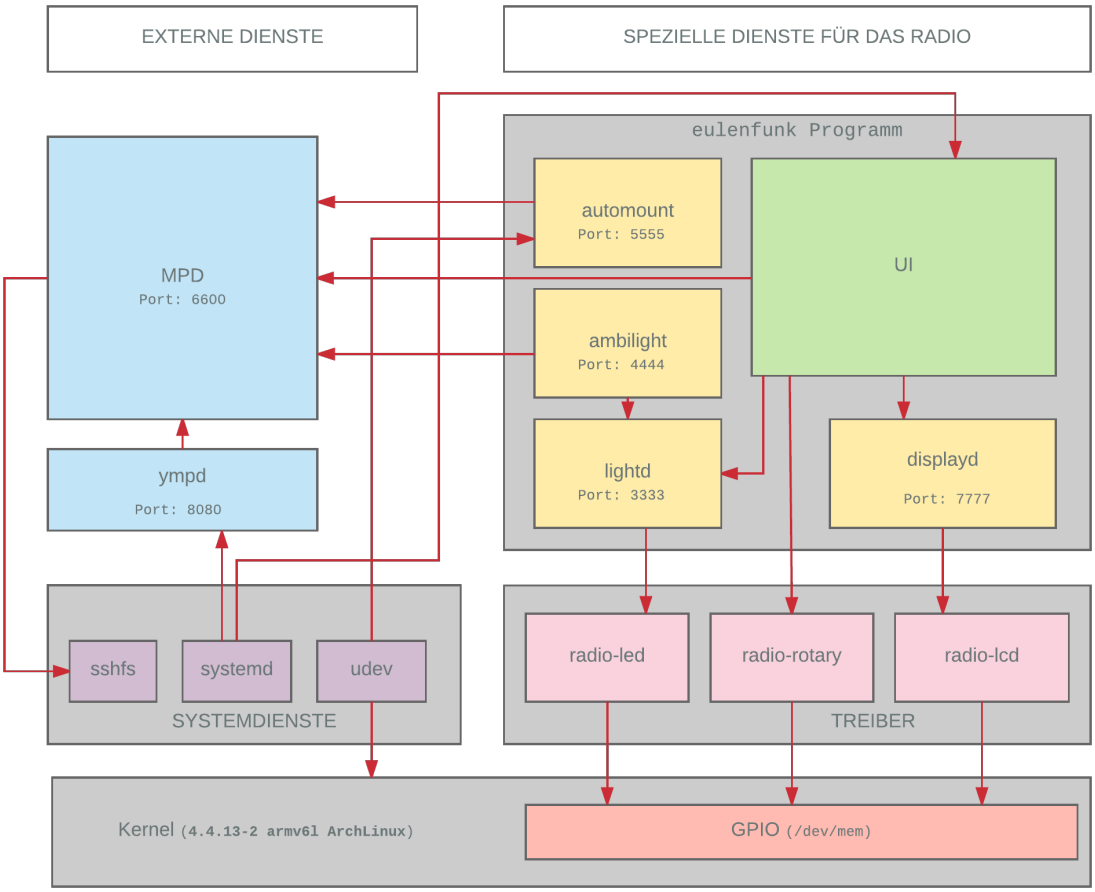

Abbildung 4.2: Übersicht über die Softwarelandschaft von *Eulenfunk*. Dienste mit einer Netzwerkschnittstelle sind durch den entsprechenden Port gekennzeichnet.

4.4.1 Vorhandene Softwarebibliotheken

`wiringPi` (http://wiringpi.com): Eine Portierung der Arduino–`Wiring` Bibliothek von Gordon Henderson auf den *Raspberry Pi*. Sie dient wie ihr Arduino–Pendant zur leichten Steuerung der verfügbaren Hardware, insbesondere der GPIO–Pins über `/dev/mem`. Daneben wird für den LCD–Treiber auch die mitgelieferte LCD–Bibliothek genutzt. Für den LED–Treiber wird zudem die softwarebasierte Pulsweitenmodulation genutzt, allerdings in einer leicht veränderten Form.

`go-mpd` (https://github.com/fhs/gompd): Eine einfache MPD–Bibliothek, die die wichtigsten Kommandos des MPD–Protokolls unterstützt.

`go-colorful` (github.com/lucasb-eyer/go-colorful): Eine Bibliothek um Farben in verschiedene Farbräume zu konvertieren. Der Dienst, der die LED passend zur Musik setzt, nutzt diese Bibliothek um RGB–Farbwerte in den HCL–Farbraum zu übersetzen. Dieser eignet sich besser um saubere Übergänge zwischen zwei Farben zu berechnen und Farbanpassungen vorzunehmen.

`cli` (github.com/urfave/cli): Eine komfortable und reichhaltige Bibliothek, um Kommandozeilenargumente zu parsen. Unterstützt Subkommandos ähnlich wie `git`, welche dann wiederum eigene Optionen oder weitere Subkommandos besitzen können. Beide Features wurden extensiv eingesetzt, um alle in *Go* geschriebenen Dienste in einer Binärdatei mit konsistentem Kommandozeileninterface zu vereinen.

4.5 Treiber–Software

In Summe gibt es momentan drei unterschiedliche Treiber. Sie finden sich im `driver/` Unterverzeichnis[2] der Software nebst einem passenden Makefile. Nach dem Kompilieren entstehen drei Binärdateien, welche mit dem Präfix `radio-` beginnen:

- `radio-led`: Setzt die Farbe des LED–Panels auf verschiedene Weise.
- `radio-lcd`: Liest Befehle von `stdin` und setzt das Display entsprechend.
- `radio-rotary`: Gibt Änderungen des Drehimpulsgebers auf `stdout` aus.

Die genaue Funktionsweise dieser drei Programme wird im Folgenden näher beleuchtet.

4.5.1 LED–Treiber (`driver/led-driver.c`)

Der LED–Treiber dient zum Setzen eines RGB–Farbwerts. Jeder Kanal hat den Wertebereich 0 bis 255. Die Hilfe des Programms zeigt die verschiedenen Aufrufmöglichkeiten:

```
$ radio-led
Usage:
  radio-led on  ....... turn on LED (white)
  radio-led off ....... turn off LED
```

[2] Siehe auf GitHub: https://github.com/studentkittens/eulenfunk/tree/master/driver

```
radio-led cat ....... read rgb tuples from stdin
radio-led rgb  r g b  Set LED color to r,g,b
radio-led hex  #RRGGBB Set LED color from hexstring
radio-led fade ...... Show a fade for debugging
```

Erklärung benötigt hierbei nur der `cat`–Modus, bei dem der Treiber zeilenweise RGB–Farbtripel auf `stdin` liest und setzt. Dieser Modus wird benutzt, um kontinuierlich Farben zu setzen ohne ständig das Treiberprogramm neu zu starten.

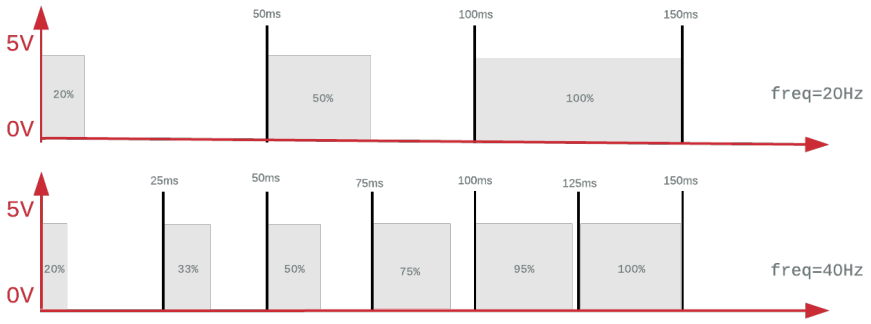

Abbildung 4.3: Grafische Darstellung der Pulsweitenmodulation mit zwei Beispielfrequenzen. Jeder weiße Block entspricht einem modulierten Wert. Die Prozentzahl darin entspricht dem »Duty–Cycle« (dt. Tastgrad). Zusammen mit dem Wertebereich ergibt sich aus ihm der eigentliche Wert. (z.B. $255 \times 0.2 = 51$)

Da ein GPIO–Pin prinzipiell nur ein oder ausgeschaltet werden kann, verwenden wir Pulsweitenmodulation (vgl. dazu [5], S 183). Dabei macht man sich die träge Natur des menschlichen Auges zu Nutze indem man die LED sehr schnell hintereinander ein– und ausschaltet. Ist die LED dabei pro Ein– und Ausschaltvorgang genauso lange hell wie dunkel, so leuchtet die LED mit etwa der Hälfte ihrer maximalen Leuchtstärke. Durch Verlängerung/Verkürzung des eingeschalteten Zustands können so viele verschiedene Helligkeitsstufen abgebildet werden. Eine beispielhafte Illustration findet sich in Abbildung 4.3

Da bei niedrigen Helligkeitswerten der ausgeschaltete Zustand besonders lange gehalten wird, kann es dazu kommen, dass ein Flackern entsteht, da man die ausgeschalteten Phasen als solche wahrnehmen kann. Um dies zu verhindern, muss eine ausreichend hohe Frequenz gewählt werden.

Anders als ursprünglich angenommen, mussten wir feststellen, dass die GPIO–Pins des *Raspberry Pi* (mit Ausnahme von Pin 18 (vgl. [5], S. 185)) kein hardwareseitiges PWM unterstützen. Aus diesem Grund mussten wir auf softwareseitiges PWM zurückgreifen, um Farben mit mindestens 256 Abstufungen zu erhalten. Nach etwas Ausprobieren befanden wir die `softPwm`–Bibliothek von `wiringPi` für tauglich.

Diese hat allerdings das Problem, dass eine hartkodierte Pulsweite von 100µs verwendet wird. Für die meisten Anwendungsfälle und den vom Autor empfohlenen 100 Abstufungen ist das auch in

Ordnung. Hundert unterschiedliche Zustände waren nach kurzem Ausprobieren bei einem weichen Farbübergang zu stark abgestuft, obwohl mit 100Hz kein nennenswertes Flackern sichtbar war:

$$T_{Periode} = 100\mu s \times 100 = 10000\mu s = 0.01s$$

$$f = \frac{1}{T_{Periode}} = 100Hz$$

Optimal wären hier 256 unterschiedliche Zustände, um die volle 8–Bit Farbtiefe auszunutzen. Daher mussten wir die entsprechende C–Datei kopieren (GPL3–lizenziert) und manuell anpassen. Dabei haben wir die Pulsweite auf 50µs herabgesetzt, was bei einer Spanne von 256 Werten eine Frequenz von optisch akzeptablen 78Hz ergibt:

$$T_{Periode} = 50\mu s \times 256 = 12800\mu s = 0.0128s$$

$$f = \frac{1}{T_{Periode}} = 78.125Hz$$

Diese Frequenz scheint optisch ausreichend flackerfrei zu sein und scheint die CPU nicht übermäßig stark zu belasten (rund +3% Last pro Farbkanal).

Es besteht eine Verbindung zu einem früheren Bastelprojekt namens `catlight`[3] — einer mehrfarbigen, in einem Gehäuse montierten LED, die über USB angesprochen werden kann. Genutzt wird diese zur Benachrichtigung bei neuen E–Mails, Chat–Nachrichten und Ähnlichem. Zu diesem Zwecke wurde auch bereits damals ein Treiberprogramm entwickelt, welches das selbe Bedienkonzept wie `radio-led` hat. Dies war während der Entwicklung von *Eulenfunk* nützlich, da es die Entwicklung der Dienste `ambilight` und `lightd` unabhängig von der Fertigstellung der Radio–Hardware machte.

4.5.2 LCD–Treiber (`driver/lcd-driver.c`)

Der LCD–Treiber setzt Bereiche des LCD–Displays auf einen gegebenen Text. Beim Start leert er das Display und liest ähnlich wie »`radio-led cat`« zeilenweise von `stdin` und entnimmt diesen Zeilen die Information welcher Bereich des Displays gesetzt werden soll. Das vom Treiber erwartete Zeilenformat ist dabei `LINENO[,OFFSET] TEXT...`, wobei `LINENO` die gewünschte Zeilennummer als Dezimalzahl ist und der optionale `OFFSET` der Index an dem geschrieben werden soll. Dahinter folgt durch ein Leerzeichen getrennt beliebiger Text. Ist kein `OFFSET` gegeben, so wird die ganze Zeile überschrieben und nötigenfalls mit Leerzeichen aufgefüllt. Ist der Text länger als die Zeile wird der Text abgeschnitten.

Der Treiber hält eine Matrix mit den aktuell gesetzten Zeichen und kann daher ein erneutes Zeichnen einer Zelle im Display verhindern, indem er das neue Zeichen mit dem Alten vergleicht. Unnötige

[3] Projektseite unter: https://github.com/studentkittens/catlight

Zeichenvorgänge waren als störende Schlieren auf dem Display wahrnehmbar.

Zudem bietet der Treiber mit dem `print-charset` Argument die Möglichkeit, die auf dem Display verfügbaren Zeichen aufs selbige auszugeben. Dazu stellt er jeweils 80 Zeichen dar und wartet einige Sekunden bevor die nächsten 80 ausgegeben werden. Hat er alle 256 Zeichen ausgegeben beendet er sich. Optional kann man auch ein Start- und End–Offset mitgeben, an dem er das Zeichnen anfangen soll.

Der Treiber unterstützt eine Reihe hardkodierter Spezialzeichen (siehe Abbildung 4.4), welche in der Menüführung und der UI benutzt werden. Das LCD unterstützt dabei 8 verschiedene *Custom Chars*, welche mittels der Codepoints 0-7 und 8-15 (wiederholt) setzbar sind. Momentan sind diese auf folgende Glyphen gesetzt:

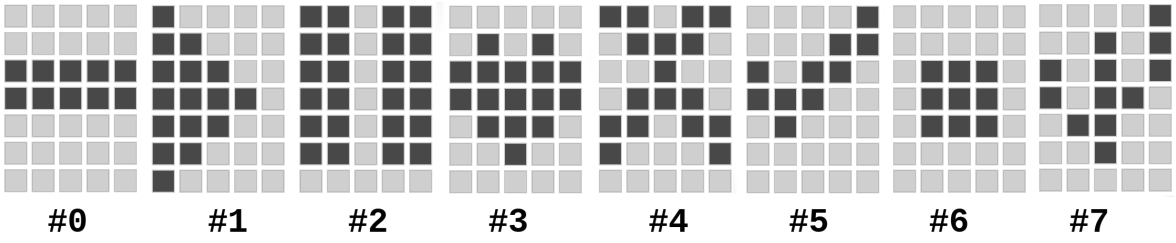

Abbildung 4.4: Spezielle, selbst gezeichnete Glyphen im Bereich 0-7 und 8-15. Gezeichnet mittels https://omerk.github.io/lcdchargen

Die eigentliche Ansteuerung der Pins übernimmt dabei wieder die `wiringPi`–Bibliothek, beziehungsweise dessen LCD–Unterbibliothek[4]. Diese ist kompatibel mit dem populären Hitachi HD44780 und Nachbauten. Das Display wird im 4–Bit Modus angesprochen. Das heißt, dass nur vier Datenpins benötigt werden (bei uns Pin 25, 24, 23, und 18).

4.5.3 Drehimpulsgeber Treiber (`driver/rot-driver.c`)

Dieser Treiber kümmert sich um das Einlesen von Werten und Ereignissen vom Drehimpulsgeber. Nach dem Start schreibt er alle registrierten Ereignisse auf `stdout`. Dabei nimmt jede Zeile ein neues Ereignis ein. Die jeweilige Zeile beginnt mit einem einzelnen Buchstaben und einem Leerzeichen, gefolgt von einem Wert. Der Buchstabe beschreibt die Art des Ereignisses:

- `v`: Änderung des **V**alues durch Drehen des Knopfes.
- `p`: Der Knopf wurde gedrückt (eine 1 folgt) oder losgelassen eine (`0` folgt).
- `t`: Der Knopf wurde für eine bestimmte Zeit gedrückt. Die Zeit folgt dahinter in Sekundenbruchteilen.

Initial wird zudem die Zeile `v 0` herausgeschrieben.

Technisch registriert sich der Treiber auf Änderungen an den GPIO–Pins 12, 13 (Pin A und B vom Drehimpulsgeber) und 14 (Button–Pin) mittels der `wiringPi`–Funtkion `wiringPiISR()`. Diese sorgt

[4] Siehe: https://projects.drogon.net/raspberry-pi/wiringpi/lcd-library

dafür, dass bei jeder Wertänderung ein Interrupt aufgerufen wird.

In der Interrupt–Routine wird der aktuelle Wert der Pins 12 und 13 ausgelesen und mit dem vorherigen Wert verglichen. Dadurch ist es möglich zu entscheiden, in welche Richtung der Drehknopf bewegt wurde ohne dass ein Prellen auftritt. Mehr Informationen zum sogenannten »Grey Code« findet sich unter [7], S. 362 und folgenden Seiten.

Da pro Einrastung des Drehknopfs ca. vier Interrupts getriggert werden, wird auf eine globale Gleitkommazahl der Wert $\frac{1}{4}$ addiert. Beim Herausgeben des Wertes wird der Wert dann auf den nächsten Integer gerundet. Auch für jeden Knopfdruck wird ein Interrupt ausgelöst. Da der Knopf prellt, wird hier mit einem niedrigen Timeout gearbeitet, welcher die Störsignale filtert.

Da in Interrupts nur reentrante Funktionen aufgerufen werden sollten, werden nur globale Flags in den Interruptfunktionen gesetzt. In der `main`–Funktion läuft eine Schleife mit einem Timeout von 50 Millisekunden, welche diese Werte abholt, formatiert und auf `stdout` schreibt. Der ursprüngliche Code für diesen Treiber stammt dabei von einem Blogpost[5]. Der Code wurde etwas aufgeräumt und um Knopfdrücke sowie Ausgabe auf `stdout` erweitert.

4.6 Service Software

Die folgenden Dienste implementieren die eigentliche Logik von *Eulenfunk*. Alle Dienste finden sich in einer gemeinsamen Binärdatei namens `eulenfunk`:

```
$ eulenfunk help
NAME:
   eulenfunk - Control the higher level eulenfunk services

USAGE:
   eulenfunk [global options] command [command options] [arguments...]

VERSION:
   0.0.1

AUTHOR(S):
   Waldsoft <sahib@online.de>

COMMANDS:
     info       Send mpd infos to the display server on the `mpd` window
     ui         Handle window rendering and input control
     automount  Control the automount for usb sticks filled with music
     lightd     Utility server to lock the led and enable nice atomic effects
     display    Display manager and utilites
```

[5] Ursprünglicher Treiber: http://theatticlight.net/posts/Reading-a-Rotary-Encoder-from-a-Raspberry-Pi

```
     ambilight  Control the ambilight feature

GLOBAL OPTIONS:
   --width value   Width of the LCD screen (default: 20) [$LCD_HEIGHT]
   --height value  Height of the LCD screen (default: 4) [$LCD_HEIGHT]
   --help, -h      show help
   --version, -v   print the version
```

Details zu den Optionen der jeweiligen Dienste können mittels `eulenfunk help <service>` angezeigt werden.

4.6.1 `displayd` – Der Displayserver

4.6.1.1 Einleitung

Der Displayserver `displayd` kümmert sich um die Verwaltung der Display–Inhalte. Er bietet eine höhere Abstraktionsschicht als der vergleichsweise simple LCD–Treiber. Dabei bietet er die Abstraktion von *Zeilen*, *Fenstern* und erleichtert dem Programmierer Enkodierungsaufgaben indem es ein Subset von Unicode unterstützt. Eine Zeile ist dabei ein beliebig langer UTF8–enkodiertert Text ohne Zeilenumbruch. Die Zeile kann dabei länger als das Display sein. In diesem Fall wird die Zeile abgeschnitten oder scrollt je nach Konfiguration mit einer bestimmten Geschwindigkeit durch. Ein Fenster hingegen ist eine benannte Ansammlung von Zeilen. Auch ein Fenster kann mehr Zeilen haben als das Display physikalisch bietet. Vom Nutzer kann der Fensterinhalt dann vertikal verschoben werden. Es können mehrere Fenster verwaltet werden, aktiv ist dabei aber nur ein Ausgewähltes.

Die Idee diese Funktionalität in einem eigenen Daemon auszulagern, ist vom Grafikstack in unixoiden Betriebssystemen inspiriert. Dabei kümmert sich ebenfalls ein Displayserver um die Verwaltung der Inhalte (meist `X.org` oder `Wayland`) indem er auf bestimmte Treiber zurückgreift. Der Nutzer kann dann mittels eines festgelegten Protokolls mit dem Displayserver sprechen und so unabhängig von der verwendeten Rendering–Methode Inhalte darstellen. Da das Protokoll zwischen Client und Displayserver meist trotzdem zu komplex ist, haben sich für diese Aufgaben simple UI–Bibliotheken wie Xlib oder komplexere GTK+ und Qt etabliert. Auch die Fenster–Metapher wurde dabei von den Fenstermanagern übernommen.

4.6.1.2 Architektur

Das Protokoll von `displayd` ist ein relativ simpel gehaltenes, zeilenbasiertes Textprotokoll. Für den Zugriff auf dasselbige wird daher auch keine UI–Bibliothek benötigt, lediglich einige Netzwerk– und Formatierungs–Hilfsfunktionen wurden implementiert[6]. Basierend auf diesen Primitiven wurden aber auf Clientseite Funktionalitäten wie Menü–»Widgets« implementiert, welche die grafische Darstellung mit der Nutzereingabe verquicken.

[6] Siehe https://godoc.org/github.com/studentkittens/eulenfunk/display#LineWriter

Neben diesen Aufgaben löst `displayd` ein architektonisches Problem: Wenn mehrere Anwendung versuchen auf das Display zu schreiben, käme ohne zentrale Instanz ein eher unleserliches Resultat dabei heraus. Durch `displayd` können Anwendungen auf ein separates Fenster schreiben, wovon jeweils nur eines aktiv angezeigt wird. Abbildung 4.5 zeigt die Architektur in der Übersicht.

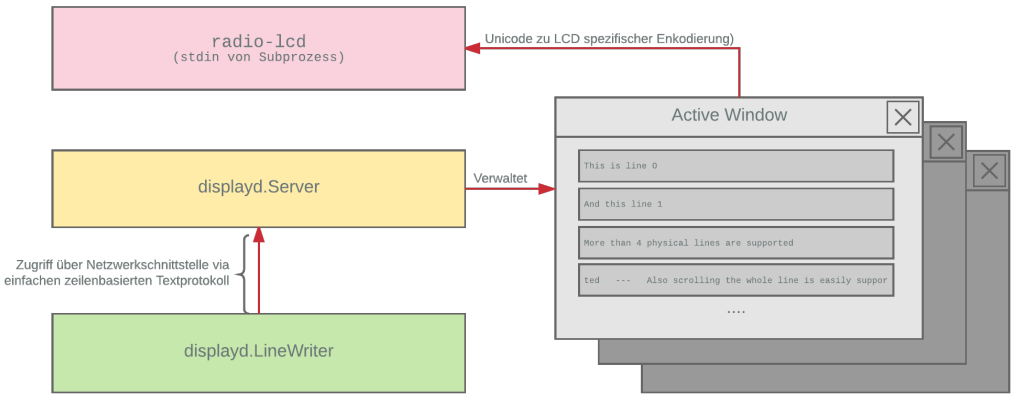

Abbildung 4.5: Architekturübersicht von `displayd` mit Beispielfenstern.

Nach dem Start kann man auf Port 7777 `displayd` mittels eines simplen, zeilenbasierten Textprotokolls kontrollieren. Dabei werden die folgenden Kommandos (mit Leerzeichen–getrennten Argumenten) unterstützt:

```
switch <win>                -- Wechsle zum Fenster namens <win>.
line <win> <pos> <text>...  -- Setze die Zeile <pos> im Fenster <win> zu <text>
scroll <win> <pos> <delay> -- Lässt Zeile <pos> in Fenster <win> mit <delay> scrollen.
move <win> <off>           -- Verschiebe Fenster <win> um <off> Zeilen nach unten.
truncate <win> <max>        -- Schneide Fenster <win> nach <max> Zeilen ab.
render                      -- Gebe aktuelles Fenster auf Verbindung aus.
close                       -- Schließt die aktuelle Verbindung.
quit                        -- Beendet Daemon und schließt Verbindung.
```

Dabei kann in die `<Platzhalter>` folgendes eingesetzt werden:

```
<win>:   Ein valider Fenstername
         (andernfalls wird ein neues Fenster mit diesen Namen angelegt)
<pos>:   Eine Zeilennummer, beginnend bei 0 für die erste Zeile.
<off>:   Ein Offset; 0 steht für keine Änderung; Kann negativ sein.
<max>:   Maximale Zeilenanzahl nach der das Fenster abgeschnitten wird.
<delay>: Zeitlicher Abstand zwischen zwei Scroll--Vorgängen.
         Siehe auch: https://golang.org/pkg/time/#ParseDuration
         Beispiel: 100ms
```

Für die tatsächliche Anzeige nutzt `displayd` wie oben erwähnt das Treiberprogramm `radio-lcd`. Dabei wird in periodischen Abständen (momentan 150ms) das aktuelle Fenster auf den Treiber geschrieben. Zukünftige Versionen sollen dabei intelligenter sein und nur die aktuell geänderten

Zeilen herausschreiben. Allerdings hat sich herausgestellt, dass man mit mehreren scrollenden Zeilen bereits mit diesem ereignisbasierten Ansatz auf eine höhere Aktualisierungsrate kommt als mit den statischen 150ms. Eine Art »VSync«, welches die Aktualisierungsrate intelligent limitiert wäre hier in Zukunft wünschenswert.

4.6.1.3 Entwicklung mit `displayd`

Da der *Raspberry Pi* nur bedingt als Entwicklungsplattform tauglich ist (langsamer Compile/Run Zyklus), unterstützt `displayd` auch Debugging–Möglichkeiten. Im Folgenden werden einige Möglichkeiten gezeigt mit `displayd` zu interagieren, beziehungsweise Programme zu untersuchen, die `displayd` benutzen:

```
# Den display server starten; --no-encoding schaltet spezielles LCD encoding
# ab welches auf normalen Terminals zu Artefakten führt.
$ eulenfunk display server --no-encoding &

# Gebe in kurzen Abständen das "mpd" Fenster aus.
# (In separaten Terminal eingeben!)
$ eulenfunk display --dump --update --window mpd

# Verbinde zu MPD und stelle aktuellen Status auf "mpd" Fenster dar.
# (auch in separaten Terminal eingeben)
$ eulenfunk info
# Auch nach Unterbrechung wird der zuletzt gesetzte Text weiterhin angezeigt:
$ <CTRL-C>
# Änderungen sind auch möglich indem man direkt mit dem Daemon über telnet
# oder netcat spricht. Hier wird die erste Zeile überschrieben, das aktuelle
# Fenster angezeigt und dann die Verbindung geschlossen.
$ telnet localhost 7777
line mpd 0 Erste Zeile geändert!
render
(... Ausgabe ...)
close
```

4.6.1.4 LCD–optimierte Enkodierung

Das LCD unterstützt 8–bit pro Zeichen. Dabei sind die ersten 127 Zeichen weitestgehend deckungsgleich mit dem ASCII–Standard. Lediglich die Zeichen 0 bis 31 sind durch *Custom Chars* und einige zusätzliche Zeichen belegt. Dies ist insofern auch sinnvoll, da in diesem Bereich bei ASCII Steuerzeichen definiert sind, die auf dem LCD schlicht keinen Effekt hätten.

Die Zeichen 128 bis 255 sind vom Hersteller des Displays mit verschiedenen Symbolen belegt worden,

die keinem dem Autor bekannten Encoding entsprechen. Da auch nach längerer Internetrecherche keine passende Encoding–Tabelle gefunden werden konnte, wurde (in mühevoller Handarbeit) eine Tabelle erstellt, die passende Unicode–Glyphen auf das jeweilige Zeichen des Displays abbildet. Nicht erkannte UTF8–Zeichen werden als ein »?« gerendert anstatt Zeichen die mehrere Bytes zur Enkodierung (wie » μ «) als mehrere falsche Glyphen darzustellen. So wird beispielsweise aus dem scharfen »ß« das Zeichen 223. Diese Konvertierung wird transparent von `displayd` vorgenommen, wodurch es möglich wird die meisten Musiktitel und Ähnliches annäherend korrekt darzustellen.

Abbildung 4.6 zeigt das erstellte Mapping zwischen Unicode und LCD–Display. Folgende Seiten waren bei der Erstellung der Tabelle hilfreich:

- http://www.amp-what.com (Suche mittels Keyword)
- http://shapecatcher.com (Suche mittels Skizze)

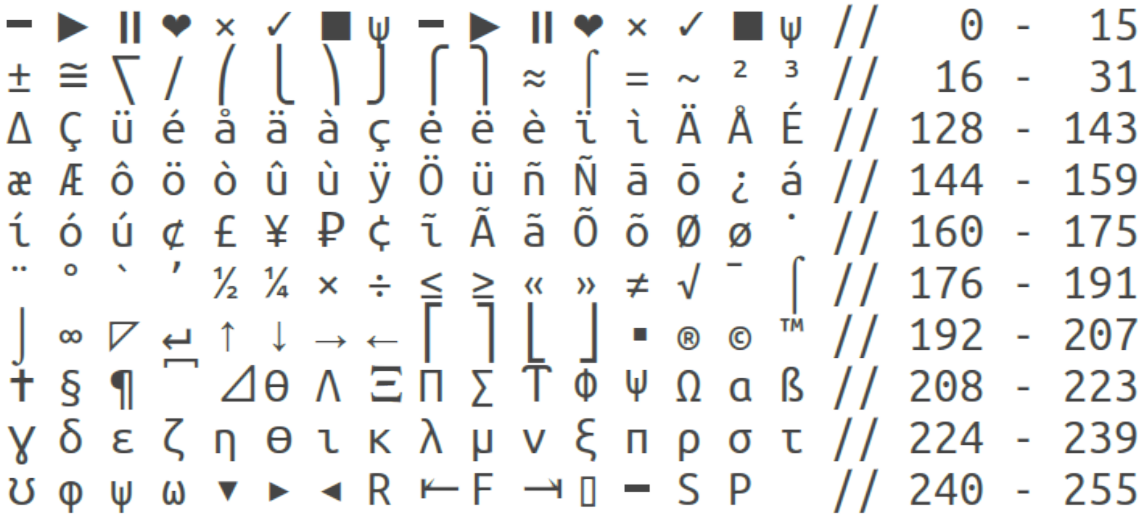

Abbildung 4.6: Unicode–Version der LCD–Glyphen. Der normale ASCII–Bereich (32-127) wurde ausgelassen.

4.6.2 `lightd` – Der Effektserver

`lightd` ist ein relativ einfacher Service, dessen Hauptaufgabe die Verwaltung des Zugriffs auf die LEDs ist. Wollen mehrere Programme die LED ansteuern, um beispielsweise einen sanften roten und grünen Fade–Effekt zu realisieren, so würde ohne Synchronisation zwangsläufig eine zittrige Mischung beider Effekte entstehen.

Ursprünglich war `light` als »`lockd`« konzipiert, der den Zugriff auf verschiedene Ressourcen verwalten kann. Da aber das Display bereits von `displayd` synchronisiert wird und durchaus mehrere Programme den Drehknopf auslesen dürfen, wurde diese etwas generellere Idee wieder verworfen.

Die zweite Hauptaufgabe von `lightd` ist die Darstellung von Farbeffekten auf der LED. Ursprünglich waren diese dafür gedacht, um beispielsweise beim Verlust der WLAN–Verbindung ein rotes Blinken anzuzeigen. Momentan wird allerdings nur beim Herunterfahren bzw. Rebooten ein rotes bzw.

oranges Blinken angezeigt. Folgende Effekte sind also momentan als Möglichkeit zur Erweiterung zu begreifen:

- `blend`: Überblendung zwischen zwei Farben.
- `flash`: Kurzes Aufblinken einer bestimmten Farbe.
- `fade`: Lineare Interpolation von schwarz zu einer bestimmten Farbe und zurück.
- `fire`: Kaminfeuerartiges Leuchten.

Wie andere Dienste wird auch `lightd` mittels einer Netzwerkschnittstelle kontrolliert. Die möglichen Kommandos sind dabei wie folgt:

```
!lock     -- Versuche exklusive Zugriffsrechte zu erlangen oder warte bis möglich.
!unlock    -- Gebe exklusive Zugriffsrechte zurück.
!close     -- Schließe die Verbindung.
<effect>   -- Auszuführender Effekt, siehe unten.
```

`<effect>` darf dabei folgendes sein:

```
{<r>,<g>,<b>}                                -- Einzelne Farbe.
blend{<src-color>|<dst-color>|<duration>}    -- Blend Effekt.
flash{<duration>|<color>|<repeat>}           -- Flash Effekt.
fire{<duration>|<color>|<repeat>}            -- Fire Effekt.
fade{<duration>|<color>|<repeat>}            -- Fade Effekt.
```

Der Platzhalter `<*-color>` steht dabei für eine einzelne Farbe. `<duration>` ist eine zeitliche Dauer und `<repeat>` eine Ganzzahl, die beschreibt wie oft der Effekt wiederholt wird.

Die Farbe wird, ähnlich wie bei `displayd`, auf `stdin` von `radio-led` geschrieben.

4.6.3 `ambilightd` – Optische Musikuntermalung

Ein »Gimmick« von *Eulenfunk* ist es, dass die LED entsprechend zur momentan spielenden Musik eingefärbt wird. Hier erklärt sich auch der Name dieses Dienstes: *Ambilight* (vgl. [16]) bezeichnet eigentlich eine von Phillips entwickelte Technologie, um an einem Fernseher angebrachte LEDs passend zum momentanen Bildinhalt einzufärben. Hierher kommt auch die ursprüngliche Idee, dies auf Musik umzumünzen.

Um aus den momentan spielenden Audiosamples eine Farbe abzuleiten, gibt es einige Möglichkeiten (vgl. Kapitel 3.12.1). Eine große Einschränkung bildet hierbei allerdings die sehr begrenzte Rechenleistung des *Raspberry Pi*. Daher haben wir uns für eine Variante entschieden, bei der die Farbwerte vorberechnet werden. Das hat den offensichtlichen Nachteil, dass man für Radiostreams kein Ambientlicht anzeigen kann. Andererseits möchte man das bei Nachrichtensendung und Diskussionsrunden vermutlich auch nicht.

Zur Vorberechnung nutzen wir dabei das `moodbar` Programm (vgl. das Paper von Gavin Wood[1]). Dieses analysiert mit Hilfe des GStreamer–Frameworks (vgl. [17]) eine Audiodatei in einem gebräuch-

lichen Format und zerlegt diese in 1000 Einzelteile. Sehr oberflächlich erklärt[7] wird für jedes dieser Teile ein Farbwert berechnet, wobei niedrige Frequenzen tendenziell zu roten Farbtönen werden, mittlere Frequenzen zu Grüntönen und hohe Frequenzen zu blauen Tönen werden. Die so gesammelten Farbwerte werden dann in einer `.mood`–Datei gespeichert, welche aus 1000 RGB–Tripeln à 3 Byte (1 Byte pro Farbkanal) bestehen. Ein visualisiertes Beispiel für eine Moodbar kann man in Abbildung 4.7 sehen.

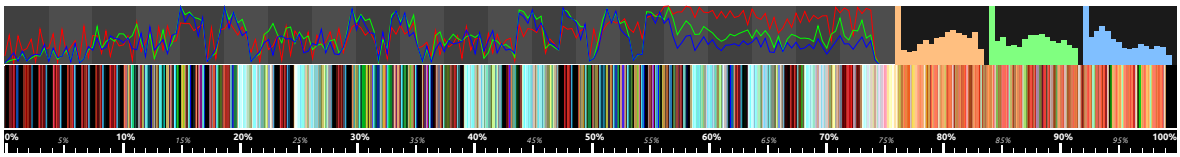

Abbildung 4.7: Moodbar–Visualisierung des Liedes »We will rock you« von »Queen«. Das Fußstapfen und Klatschen am Anfang ist gut erkennbar. Die Visualisierung wurde mit einem Python–Skript des Autor erstellt (https://github.com/sahib/libmunin/blob/master/munin/scripts/moodbar_visualizer.py)

Um nun aber tatsächlich ein zur Musik passendes Licht anzuzeigen, muss für jedes Lied in der Musikdatenbank eine passende Moodbar in einer Datenbank abgespeichert werden. Diese Datenbank ist im Fall von *Eulenfunk* ein simples Verzeichnis in dem für jedes Lied die entsprechende Moodbar mit dem Pfad relativ zum Musikverzeichnis[8] als Dateinamen abgespeichert wird.

Die Datenbank kann dabei mit folgendem Befehl angelegt und aktuell gehalten werden:

```
$ eulenfunk ambilight --update-mood-db --music-dir /music --mood-dir /var/mood
```

Die eigentliche Aufgabe von `ambilightd` ist es nun den Status von MPD zu holen, die passende `.mood`–Datei zu laden und anhand der Liedlänge und der bereits vergangenen Zeit den aktuell passenden Sample aus den 1000 vorhandenen anzuzeigen. Damit der Übergang zwischen den Samples flüssig ist wird linear zwischen den einzelnen Farbwerten überblendet. Da die LED eher zu einer weißen Farbe tendiert, wenn mehrere Kanäle an sind, werden mittlere Sättigungwerte leicht verstärkt und mittlere Helligkeitswerte etwas abgeschwächt.

Auch `ambilight` ist über Netzwerk über ein zeilenbasiertes Textprotokoll auf Port 4444 ansprechbar. Es werden die Kommandos `off` (Halte `ambilight` Ausgabe an), on (Führe `ambilight` Ausgabe fort) und `state` (Zeige aktuellen Status) unterstützt. Eine Übersicht über alle beteiligten Komponenten von `ambilight` findet sich in Abbildung 4.8.

Im Fazit kann man sagen, dass die verwendete Technik durchaus gut für die meisten Lieder funktioniert. Besonders die Synchronisation ist dabei erstaunlich akkurat und solange nur ein einzelnes Instrument spielt, wird dem auch eine passende Farbe zugeordnet. Ein Dudelsack erscheint beispielsweise meist grün, während ein Kontrabass in mehreren Liedern rot erschien.

Lediglich bei schnellen Tempowechseln (Beispiel: »Prison Song« von »System of a Down«) sieht man, dass der Farbübergang bereits anfängt bevor man den zugehörigen Ton hört. Dem könnte im Zukunft

[7] Der eigentliche Algorithmus ist komplexer und wird im referenzierten Paper beschrieben.
[8] Wobei »/« durch »|« im Dateinamen ersetzt werden.

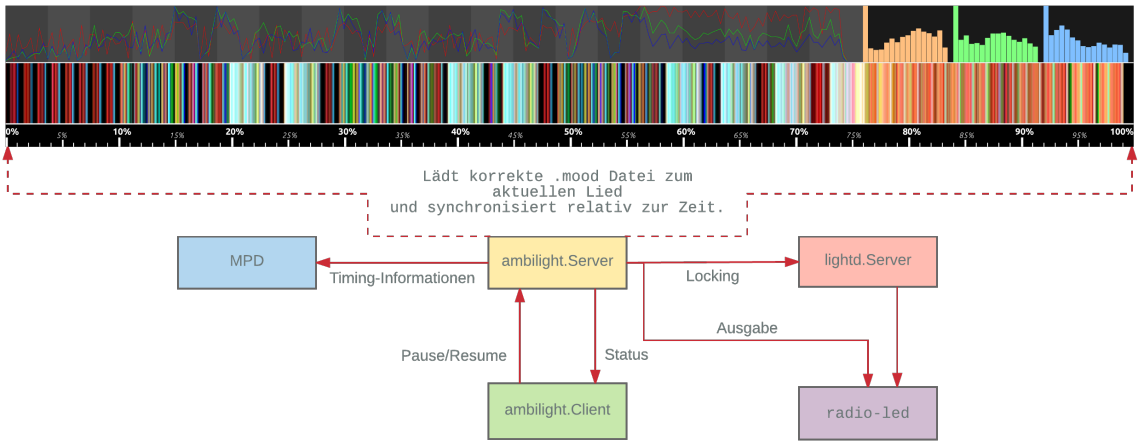

Abbildung 4.8: Konzeptuelle Übersicht über *ambilight* und verwandte Dienste.

abgeholfen werden, indem keine lineare Interpolation zwischen den Farben genutzt wird, sondern ein Verfahren, das plötzliche Übergänge eher berücksichtigt.

4.6.4 `automount` – Playlists von USB Sticks erstellen

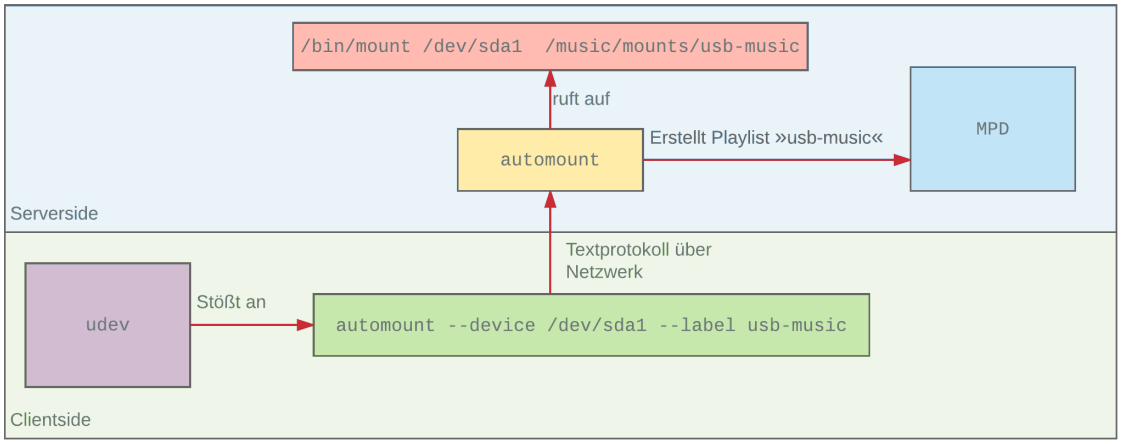

Abbildung 4.9: Beispielhafte Situation bei Anschluß eines USB Sticks mit dem Label »usb-music«

Der `autmount`–Daemon sorgt dafür, dass angesteckte USB–Sticks automatisch gemounted werden, Musikdateien darauf indiziert werden und in einer Playlist mit dem »Label« des Sticks landen.

Unter Linux kümmert sich udev um die Verwaltung des `/dev`–Verzeichnis. Zur Steuerung und Konfiguration erlaubt udev das Anlegen von Regeln. Wird ein neues Gerät angesteckt, so geht udev alle bekannten und auf das Gerät passende Regeln durch und wendet die darin definierten Aktionen an. Bei *Eulenfunk* wird die Regel `config/udev/11-music-usb-mount.rules` in das Verzeichnis `/etc/udev/rules.d` kopiert. Die `11` im Namen sorgt dafur, dass die Regel alphabetisch vor den Standardregeln abgearbeitet wird (beginnend mit `50`). Abbildung 4.9 zeigt danach den Ablauf beim

Anstecken eines USB–Sticks mit dem Label »usb-music«. Es wird von der Regel aus ein Befehl über Netzwerk an `automount` gesendet, welcher dann das Mount und Unmount samt Erstellen der Playlist übernimmt. Die genaue Erklärung der Einzelheiten wird hier aus Platzgründen ausgelassen. Weitere Informationen zu udev–Regeln sind online[9][10] zu finden.

Eine berechtigte Frage ist warum `automount` das Mounten/Unmounten des Sticks übernimmt, wenn diese Aktionen prinzipiell auch direkt von der udev–Regel getriggert werden können. Der Grund für diese Entscheidung liegt am *Namespace*–Feature von `systemd` (weit verbreitetes Init–System[11]): Dabei können einzelne Prozesse in einer Sandbox laufen, der nur begrenzten Zugriff auf seine Umgebung hat. Ruft man `/bin/mount` direkt aus der Regel heraus auf, so wird der Mount im *Namespace* von udev erstellt und taucht daher nicht im normalen Dateisystem auf. Sendet man hingegen einen Befehl an `automount`, so agiert dieser außerhalb einer Sandbox und kann den Stick ganz normal als `root`–Nutzer mounten.

Beim Entfernen des USB–Sticks wird die inverse Operation ausgeführt: Die Playlist wird gelöscht (da MPD die Lieder nicht mehr abspielen könnte) und der Mount wird wieder entfernt.

Ähnlich wie die vorherigen Dienste unterstützt `automount` einige wenige Befehle, die es über einen Netzwerk–Socket auf Port 5555 erhält:

- `mount <device> <label>`: Mounte `<device>` (z.B. `/dev/sda1`) zu `<music_dir>/mounts/<label>` und erstelle eine Playlist names `<label>` aus den Liedern.
- `unmount <device> <label>`: Entferne Playlist und Mountpoint wieder.
- `close`: Trenne die Verbindung.
- `quit`: Trenne die Verbindung und beende den Daemon.

4.6.5 `ui` – Menübasierte Bedienoberfläche

Die `ui` ist der einzige Dienst ohne Netzwerkschnittstelle. Er kümmert sich um das Anlegen und Befüllen aller Fenster und um die Steuerung fast aller anderen Dienste mittels einer menübasierten Oberfläche. Die genaue Beschaffenheit der Oberfläche wird im nächsten Kapitel (siehe Kapitel 5) im Stile eines Benutzerhandbuches beleuchtet. Daher wird hier nur ein kurzer Überblick über die Technik dahinter gegeben.

Im momentanen Zustand existieren folgende Fenster:

- `mpd`: Zeigt Infos über das momentan laufende Lied oder den aktuellen Radiostream.
- `clock`: Zeigt das aktuelle Datum und Uhrzeit.
- `stats`: Zeigt Statistiken über die MPD–Datenbank an.
- `sysinfo`: Zeigt die Ausgabe des Skripts `config/scripts/radio-sysinfo.sh` an.
- `about`: Zeigt die Credits an.
- `menu-main` Hauptmenü von dem aus alle Funktionen erreichbar sind.

[9] Writing udev rules: http://www.reactivated.net/writing_udev_rules.html
[10] Arch Linux udev: https://wiki.archlinux.de/title/Udev#Unter_.2Fmedia_einbinden.3B_Partitions_Label_verwenden_falls_vorhanden
[11] https://de.wikipedia.org/wiki/Systemd

- `menu-playlists`: Zeigt alle verfügbaren Playlists.
- `menu-power`: Einträge zum Herunterfahren und Rebooten.

Fenster, die mit »`menu-`« beginnen, können durch Benutzung des Drehimpulsgebers erkundet werden. Eine Drehung nach rechts verschiebt das Menü nach unten, eine Drehung nach links nach oben.

Wie oben bereits erwähnt wurde ein kleines Client–seitiges »Toolkit« implementiert, welches die leichte Erstellung von Menüs und die Verknüpfung mit dem Drehimpulsgeber mittels Aktionen möglich macht. Der prinzipielle Aufbau dieses Toolkits ist in Abbildung 4.10 gezeigt.

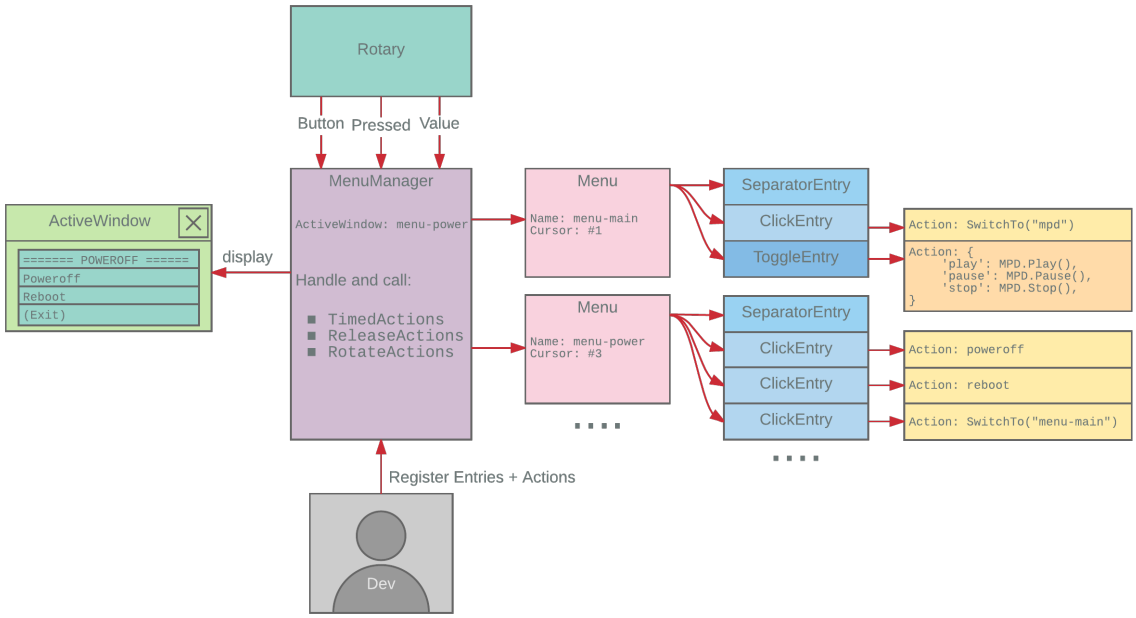

Abbildung 4.10: Aufbau der Menüstruktur aus Entwicklersicht. Der Entwickler interagiert mit "MenuManager"; dieser kümmert sich um die Ereignisverarbeitung.

Der Programmierer muss dabei dem `MenuManager` nach dessen Anlegen lediglich eine Liste von Einträgen mitgeben, die dieser verwalten soll. Es gibt momentan drei verschiedene Arten von Einträgen:

- `SeparatorEntry`: Zeigt einen vordefinierten Text. Wird zum Abtrennen verschiedener Sektionen innerhalb eines Menüs benutzt, daher der Name.
- `ClickEntry`: Zeigt einen Text und ist mit einer Aktion verknüpft. Drückt der Nutzer den Drehknopf während der Fokus auf dem Eintrag ist, so wird die Aktion ausgeführt.
- `ToggleEntry`: Wie `ClickEntry`, hat aber mehrere Zustände, die in eckigen Klammern hinter dem Text angezeigt werden. Ein Knopfdruck führt zum Weiterschalten zum nächsten Zustand. Dabei ist jeder Zustand mit einer Aktion verknüpft, die beim Umschalten ausgeführt wird.

Diese Einträge kann der Entwickler dann mit beliebigen Aktionen verknüpfen. Daneben gibt es auch noch drei andere Aktionstypen, die unabhängig vom aktuellen Eintrag ausgeführt werden:

- `TimedActions`: Wird der Knopf für eine bestimmte Zeit gehalten, kann nach einer bestimmten Zeit einmalig eine Aktion ausgeführt werden. Die Zeit nach der das passiert wird beim

Registrieren der Aktion angegeben.

- `ReleaseActions`: Wird ausgeführt wenn der Knopf gedrückt wurde, aber kein Menüeintrag ausgeführt wurde.
- `RotateActions`: Wird ausgeführt sobald der Knopf nach rechts oder links gedreht wurde.

Auf Basis dieses minimalen Toolkits wurde dann eine leicht erweiterbare Menüführung entwickelt. Eine genauere API–Beschreibung kann unter `godoc.org`[12] eingesehen werden.

4.6.5.1 `info` – Anzeige des `mpd`–Status

`info` ist der Teil der UI, welcher den Inhalt des `mpd` Fensters pflegt und darstellt. Im Hintergrund steht dabei ein voll funktionsfähiger MPD–Client, welcher auch auf Zustandsänderungen von außen reagiert. Das heißt: Ändert man das aktuelle Lied mittels eines anderen MPD–Clients (von einem Handy als Fernbedienung etwa), so wird die Änderung umgehend an die UI propagiert.

`info` kann aus Gründen der Fehlersuche auch separat von der UI gestartet werden:

```
$ eulenfunk info
```

4.6.6 `ympd` – MPD im Webbrowser

`ympd`[13] ist ein relativ populärer, in C geschriebener MPD–Client, der als Webserver fungiert und eine Bedienung von MPD im Browser via Websocket möglich macht. Auf *Eulenfunk* läuft er auf Port 8080 und kann von außen zugegriffen werden. Abbildung 4.11 zeigt die Weboberfläche.

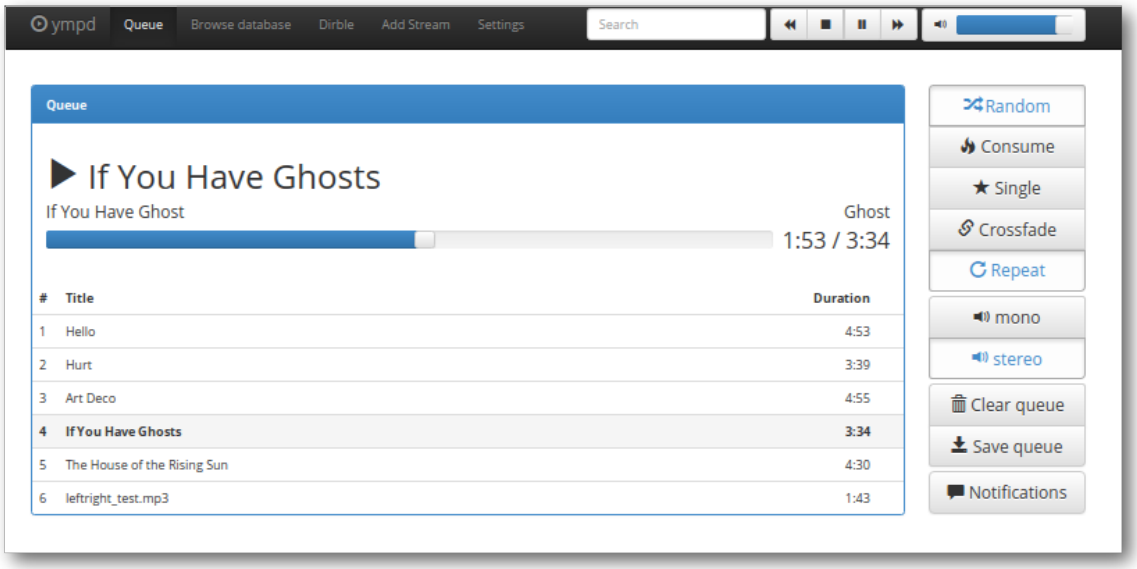

Abbildung 4.11: Screenshot der ympd–Oberfläche mit laufenden Lied aus der Testdatenbank.

[12] https://godoc.org/github.com/studentkittens/eulenfunk/ui
[13] Homepage von ympd: https://github.com/notandy/ympd

Ursprünglich war der Einsatz eines, in einer früheren Studienarbeit entwickelten, Webclients namens »Snøbær« (siehe Abbildung 4.12) angedacht. Dieser hat ein paar mehr Features wie das automatische Herunterladen von Coverart und Liedtexten. Leider lies sich dieser nicht ohne größeren Aufwand auf dem *Raspberry Pi* kompilieren, weswegen aus Zeitgründen einstweilen auf `ympd` umgeschwenkt wurde.

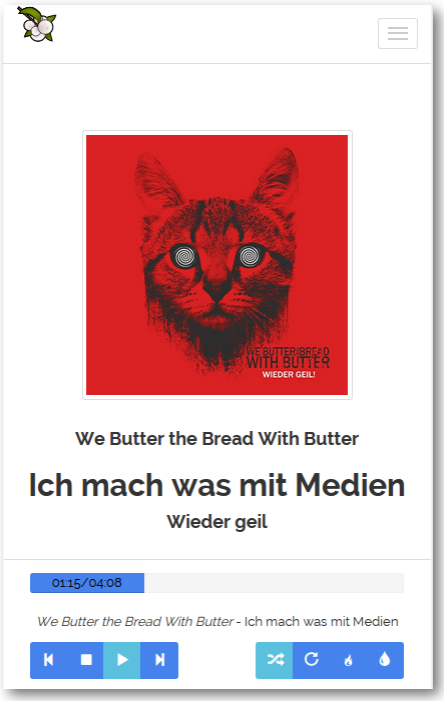

Abbildung 4.12: Screenshot von Snøbær's Weboberfläche.

4.6.6.1 Zeroconf

Normalerweise ist die Weboberfläche von *Eulenfunk* unter der Adresse `http://eulenfunk:8080` erreichbar. Falls das wegen mangelnder Namensauflösung aber nicht funktioniert, kann man *Zeroconf* (vgl. [18]) dazu nutzen die IP von *Eulenfunk* herauszufinden:

```
$ avahi-browse _mpd._tcp -r | grep 'hostname = \[eulenfunk' -A 2
   hostname = [eulenfunk.local]
   address = [192.168.23.30]
   port = [6600]
```

Das funktioniert, weil der MPD–Server seine Anwesenheit mittels Zeroconf–Protokolls mitteilt. Es muss allerdings der Avahi–Daemon sowohl auf dem anfragenden Rechner als auch auf *Eulenfunk* aktiv sein.

4.7 Einrichtung

In diesem Teilkapitel soll die Einrichtung aller relevanten Komponenten dokumentiert werden. Dies soll hauptsächlich zur Referenz dienen, um das Radio nachbauen zu können.

4.7.1 mpd und ympd

Installation aus den offiziellen Quellen (mpd) und aus dem AUR (ympd):

```
$ pacman -S mpd mpc
$ yaourt -S ympd
$ mkdir -p /var/mpd/playlists
$ touch /var/mpd/mpd.{db,state,log}
$ cp eulenfunk/config/mpd.conf /var/mpd
$ mpd /var/mpd/mpd.conf
$ mpc update -w
```

mpd und ympd sind die einzigen Dienste, die von Außen (ohne Authentifizierung) zugreifbar sind. Auch wenn *Eulenfunk* normal in einem abgesicherten WLAN hängt, wurde für die beiden Dienste jeweils ein eigener Nutzer mit eingeschränkten Rechten und `/bin/false` als Login–Shell angelegt.

Im täglichen Betrieb wird die Musik von einem anderen Rechner via `sshfs` eingehängt. Dazu wurde in `/etc/fstab` ein Eintrag hinzugefügt[14], der in Verbindung mit `systemd` bewirkt, dass beim Zugriff auf das Musikverzeichnis der Mount automatisch eingehängt wird.

4.7.2 systemd – Start und Status von Diensten

`systemd` ist ein sehr mächtiges Init–System welches zum Starten und Überwachen aller Dienste in *Eulenfunk* eingesetzt wird. Im Gegensatz zu anderen Init–Systemen werden keine Shell–Skripte zum Starten genutzt, sondern sogenannte Unit–Files. Diese regeln welche Binaries gestartet werden, von was diese abhängen und was bei einem Crash passieren soll. Diese Dateien kopiert man in ein von `systemd` überwachtes Verzeichnis (beispielsweise `/usr/lib/systemd/system`). Dort kann man nach einem »`systemctl daemon-reload`« den Dienst starten und für den nächsten Bootvorgang vormerken:

```
$ systemctl start my-unit-file    # Jetzt den Dienst starten.
$ systemctl enable my-unit-file   # Beim nächsten Boot starten.
```

In Abbildung 4.13 ist schematisch der Abhängigkeitsgraph der einzelnen Dienste von *Eulenfunk* gezeigt. Jeder relevante Dienst hat dabei ein eigenes `.unit`–File[15].

[14] Siehe dazu: https://wiki.archlinux.org/index.php/SSHFS#On_demand
[15] Siehe GitHub für eine Auflistung aller Unit–Files: https://github.com/studentkittens/eulenfunk/tree/master/config/systemd

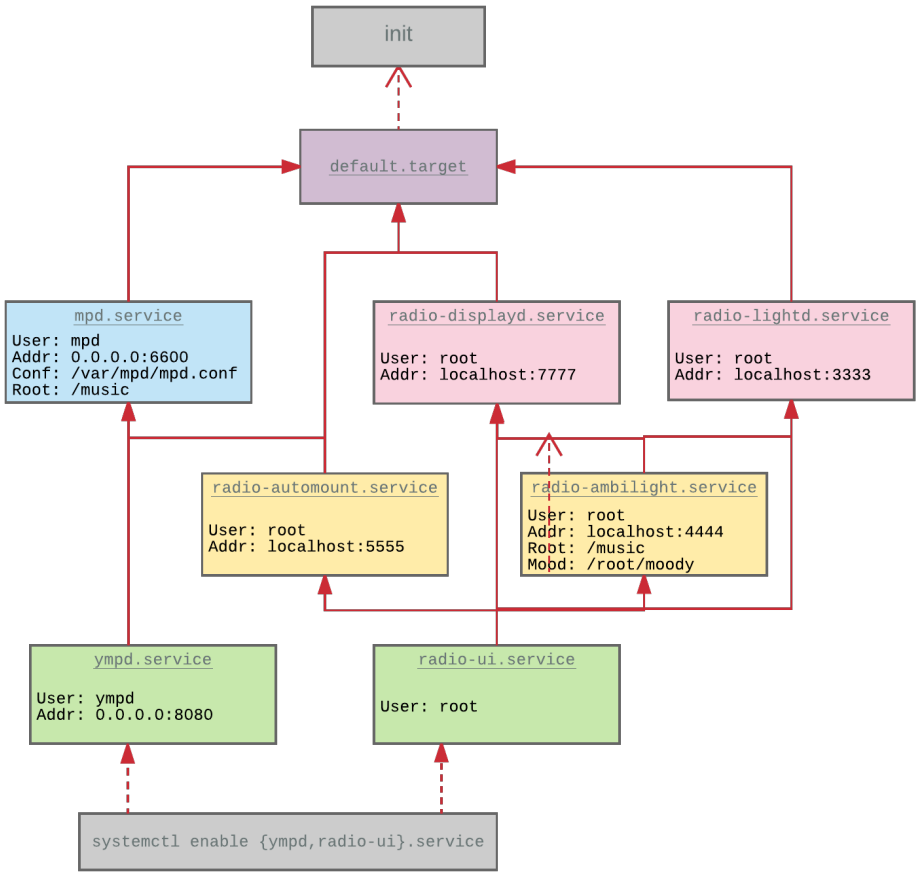

Abbildung 4.13: Abhängigkeits–Graph beim Start der einzelnen Dienste

Sollte ein Dienst abstürzen, weil beispielsweise die Software doch noch Fehler hat, dann wird der Service von `systemd` automatisch neu gestartet, da die Option `Restart=on-failure` in den `.unit`-Files von *Eulenfunk* gesetzt ist. Besonders zur Fehlersuche war `systemd` bereits sehr hilfreich, da es möglich, ist die Ausgabe des Dienstes im Nachhinein zu betrachten:

```
$ systemctl status radio-ui.service
* radio-ui.service - MPD client that show the current state on the LCD
  Active: active (running) since Mon 2016-06-27 23:45:55 CEST; 2h 0min ago
[...]
Jun 28 01:19:24 eulenfunk eulenfunk[410]: 2016/06/28 01:19:24 Pressed for 1s
[...]
# Alternativ falls mehrere Dienste zeitgleich angezeigt werden sollen:
$ journalctl -u radio-ambilight.service -u radio-ui.service --since today
[...]
```

4.7.3 eulenfunk Software

Die Software selbst kann ohne große Abhängigkeiten direkt von *GitHub* installiert werden:

```
$ pacman -S wiringpi go git
$ git clone https://github.com/studentkittens/eulenfunk && cd eulenfunk
$ mkdir -p /root/go
$ export GOPATH=/root/go
$ export GOBIN=/root/go/bin
$ go get .
$ make install
```

Das mitgelieferte Makefile installiert alle `.unit`–Files, Skripte, Treiber–Binärdateien und die `eulenfunk`–Binärdatei selbst. Die Software sollte dabei auch auf normalen Desktop–Rechnern kompilierbar sein, ist dort aber mangels passender Treiber nur bedingt ausführbar.

4.8 Fazit

Die Anforderungen können durchaus als erfüllt betrachtet werden. Anforderung **#3** *(Ausfallsicherheit)* und **#4** *(Wartbarkeit)* wurden durch den konsequenten Einsatz von `systemd` umgesetzt. Anforderung **#5** *(Lose Kopplung)* wird durch die Aufteilung der Dienste in einzelne, durch das Netzwerk getrennte, Prozesse erreicht. Die Erweiterbarkeit sollte dadurch gewährleistet sein, dass relativ klar strukturierter und modularer Code verfasst wurde. Möchte man sich mit dem Code vertraut machen, so hilft die API–Dokumentation[16] weiter.

Anforderung **#2** *(Effizienz)* wurde durch die Wahl effizienter Programmiersprachen und Vermeidung ressourcenhungriger oder ineffizienter Programmierung vermieden. Die CPU–Last bewegt sich dabei meist zwischen 40–60% bei `mp3`–enkodierten Liedern. Bei `.flac`–Dateien liegt die Last etwa 10–20 Prozentpunkte höher. Die Speicherauslastung ist auch nach mehreren Stunden Benutzung bei konstanten 50MB.

Auch wurde versucht, die Anzahl von gestarteten Prozess klein zu halten und nur das Nötigste zu starten:

```
$ pstree -A
systemd-+-2*[agetty]
        |-avahi-daemon---avahi-daemon
        |-dbus-daemon
        |-dhcpcd
        |-eulenfunk display-+-radio-lcd
        |-eulenfunk lightd-+-radio-led---3*[{radio-led}]
        |-eulenfunk ambilight-+-2*[radio-led---3*[{radio-led}]]
        |-eulenfunk automount---6*[{eulenfunk}]
```

[16] API–Doc Eulenfunk: https://godoc.org/github.com/studentkittens/eulenfunk

```
|-eulenfunk ui-+-radio-rotary
|              |-radio-sysinfo.sh
|-haveged
|-mpd
|-sshd---sshd---sshd---bash---pstree
|-systemd---(sd-pam)
|-systemd-journal
|-systemd-logind
|-systemd-resolve
|-systemd-timesyn---{sd-resolve) S 1
|-systemd-udevd
|-wpa_supplicant
`-ympd
```

Inwiefern Anforderung **#1** (*leichte Bedienbarkeit*) gewährleistet ist, wird das nächste Kapitel zeigen. Die Software darunter versucht die Möglichkeiten dafür zu schaffen.

4.8.1 Quelltextumfang

Mit dem Tool `cloc` (https://github.com/AlDanial/cloc) wurde eine Statistik erstellt, welche den Umfang der Software zeigt. Diese Statistik wurde bereits von auto-generierten Code und Fremdcode bereinigt:

Language	Files	Blank	Comment	Code
Go	32	991	550	4090
C	3	99	15	493
make	3	13	1	31
Bourne Shell	1	2	1	12
SUM:	39	1105	567	4626

4.8.2 Probleme und Verbesserungsmöglichkeiten

Obwohl die Software für den *Eulenfunk*-Prototypen bisher durchaus gut und stabil funktioniert, gibt es natürlich noch Verbesserungspotenzial:

- Steckt ein USB-Stick nach einem Reboot noch am Radio, so wird dieser nicht automatisch gemounted. Erst nach einem An- und Abstecken desselben ist die zugehörige Playlist wieder abspielbar. Beim Start müssten daher manuell udev-Ereignisse getriggert werden. Versuche dies zu erreichen schlugen aber leider aus bisher ungeklärten Gründen fehl.
- In seltenen Fällen verschwindet der Cursor in Menüs aus ebenfalls ungeklärten Gründen.
- Wie im Kapitel von `displayd` beschrieben, wäre auf lange Dauer ein Ereignis-basierter Ansatz im Display-Server wünschenswert, um den Stromverbrauch im Ruhezustand zu senken.

- *Eulenfunk* benötigt zum Hochfahren momentan etwa 34 Sekunden. Das ist für die meisten Anwendungsfälle vollkommen ausreichend, könnte aber eventuell noch weiter optimiert werden. Eine genaue Übersicht darüber, welche Dienste wie lang zum Start brauchen, liefert das Tool `systemd-analyze plot`. Der aktuelle Plot kann online auf GitHub eingesehen werden[17].

4.8.3 Wettervorhersage

Eine praktische Erweiterung von *Eulenfunk* wäre die Anzeige der Wettervorhersage für den aktuellen und die nächsten Tage. Oft wäre eine entsprechende Information nützlich, um beispielsweise die Durchführung einer Radtour zu planen oder ob man trockenen Fußes einkaufen gehen kann. Die Daten könnte dabei von Online–APIs wie OpenWeatherMap[18] geholt werden. Der Zugang ist dort kostenlos, aber auf 60 Anfragen pro Minute limitiert.

[17] https://github.com/studentkittens/eulenfunk/blob/master/docs/paper/images/boot-plot.svg
[18] Mehr Informationen unter: http://www.openweathermap.org/

5 Bedienkonzept/Menüsteuerung

5.1 Anforderungen

Die Eingabe–Peripherie soll möglichst einfach gehalten werden, um eine *schöne* Produkt–Optik zu gewährleisten, dabei sollen folgende Anforderungen erfüllt werden:

- Minimale sowie ansprechende Bedienelemente
- Funktionales, zweckgebundenes *Design*
- *Retro–Look* wünschenswert

Das *Design* soll im Grunde *minimalistisch* gehalten werden, das heißt, es sollen aufgrund der Übersichtlichkeit nur so wenige »Bedienelemente« wie nötig angebracht werden.

5.2 Bedienelemente

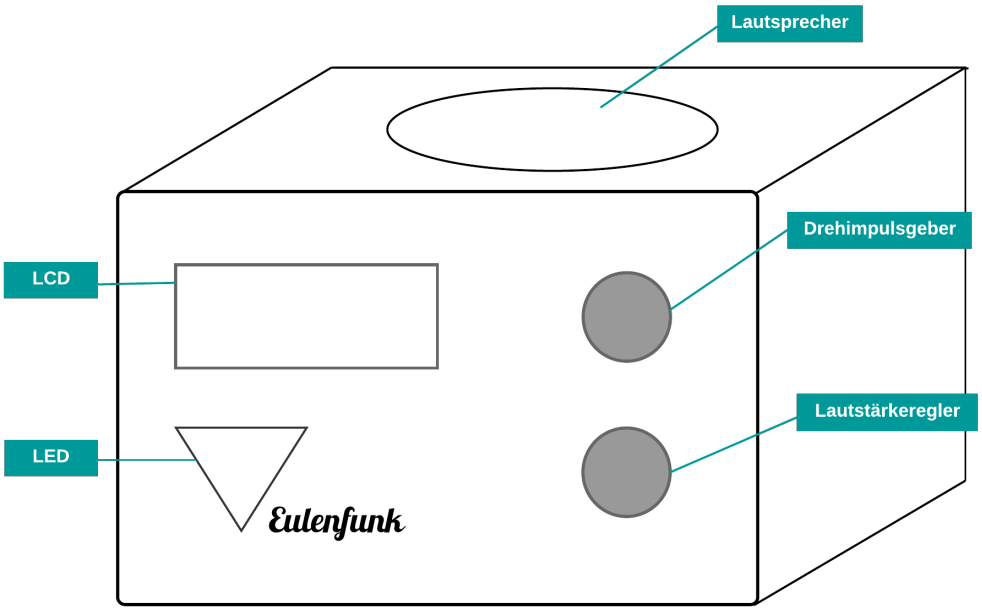

Abbildung 5.1: Frontansicht von *Eulenfunk*

Bei der Bedienung von *Eulenfunk* bestand die Herausforderung vor allem darin, trotz minimaler Anzahl an Bedienelementen eine gute Bedienbarkeit (Usability) zu erreichen.

Zur Verfügung stehen primär ein Lautstärkeregler und ein Drehimpulsgeber (Abb. 5.1). Auf der Rückseite des Radios ermöglichen zusätzlich zwei Potentiometer die Regelung der Beleuchtung und des Kontrastes vom Display. Mit einem Kippschalter kann die Audioausgabe zwischen internen und externen Lautsprechern geschaltet werden.

Wie allgemein üblich, erhöht ein »nach rechts Drehen« des Lautstärkereglers die Lautstärke und »ein nach links Drehen« verringert die Lautstärke.

Der Drehimpulsgeber (im Weiteren Master-Regler genannt) ist für die gesamte Navigation innerhalb des Menüs und das Ausführen von Aktionen zuständig. Drehen bewirkt grundsätzlich ein *Vor* oder *Zurück*. Drücken bewirkt das kontextbezogene Ausführen einer Aktion. Näheres wird bei den jeweiligen Menüansichten beschrieben.

5.3 Menüinhalt

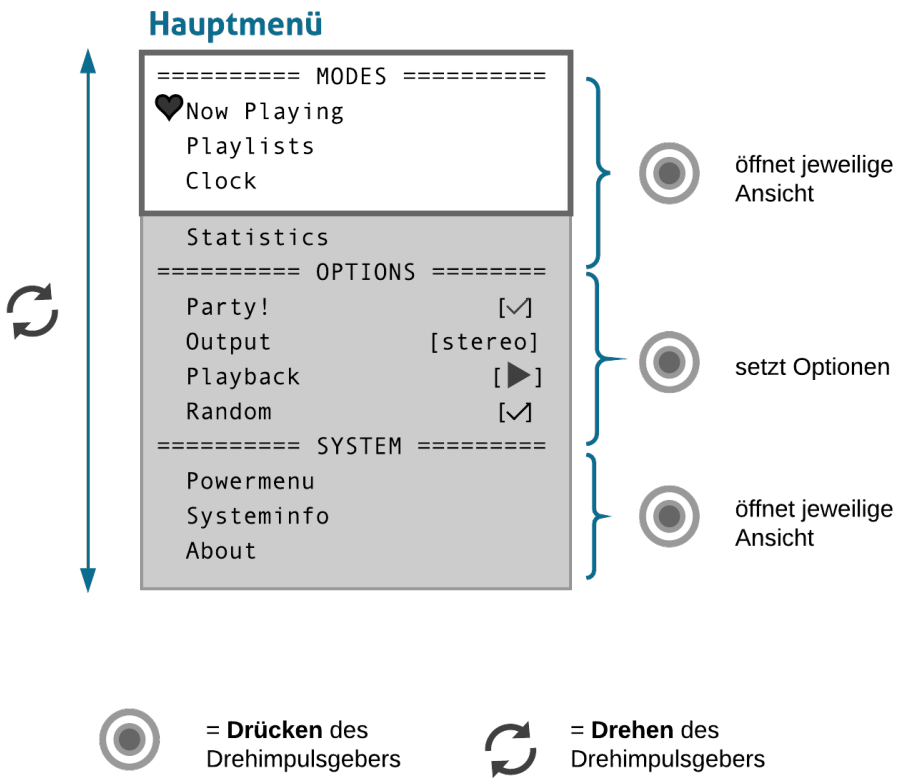

Abbildung 5.2: Ansicht des Hauptmenüs

Das Hauptmenü enthält drei Kategorien: Modes, Options und System (siehe Abb. 5.2). Ein Drehen des Master-Regler navigiert zwischen den einzelnen Einträgen. Ein Drücken des Master-Regler öffnet den gewählten Eintrag als neue Ansicht (Modes, System) oder setzt direkt andere Werte (Options).

5.3.1 Now Playing

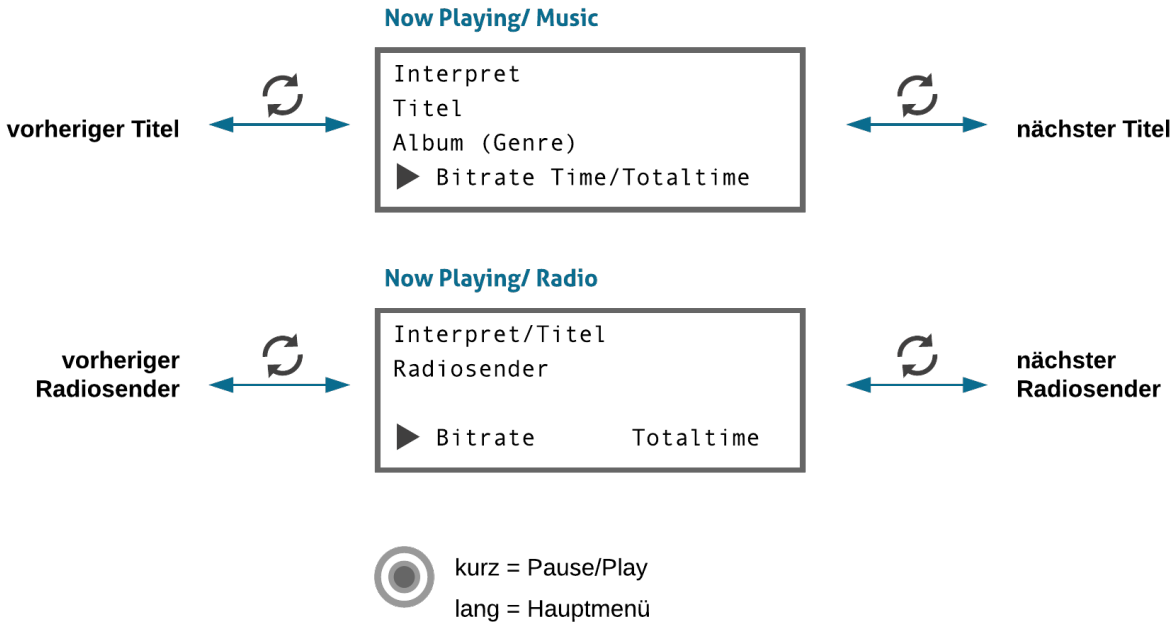

Abbildung 5.3: Ansicht des Menüpunkts *Now Playing* für Musikstücke und Radiosender

Der Menüpunkt *Now Playing* zeigt Informationen des aktuell gespielten Mediums an. Es gibt eine geringfügig abweichende Ansicht für Musikstücke und für Radiosender. Wie in Abbildung 5.3 zu sehen ist, wird bei einem Musikstück der Interpret, der Titel und das Album angezeigt. Sollte die Breite des Displays für die Länge dieser Werte nicht ausreichend sein, »scrollt« die Anzeige der Zeichen von rechts nach links durch. Die letzte Displayzeile zeigt an, ob Musik spielt, oder pausiert ist. Außerdem wird die Bitrate und die Spieldauer angezeigt. Die Ansicht für Radiosender enthält den Interpreten und den Titel in Zeile eins, gefolgt vom Radiosender in Zeile zwei.

In beiden Ansichten führt ein Drücken des Master-Reglers zum Pausieren bzw. Abspielen der Musik. Einen Titel bzw. Radiosender vor oder zurück wechseln ist durch Drehen des Reglers möglich.

5.3.2 Playlists

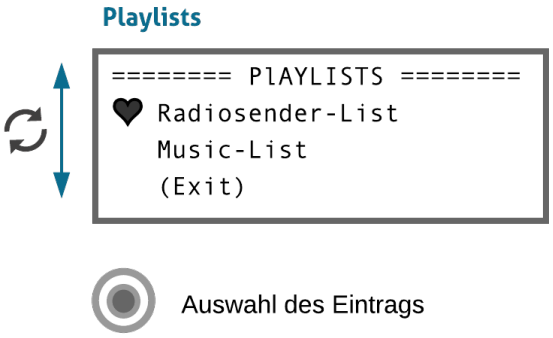

Abbildung 5.4: Ansicht des Menüpunkts *Playlists*

Der Menüpunkt *Playlists* (siehe Abb. 5.4) zeigt eine Übersicht der aktuell verfügbaren Medien. Mögliche Listen sind Radiosender und Musikstücke. Wird ein USB-Gerät an das Radio angeschlossen, erscheint er hier als Playlist.

Ein Drücken des Master-Reglers führt zur Auswahl einer Playlist. Anschließend wechselt die Anzeige auf den Modus *Current Song*, d.h. die Informationen zum aktuell aus der Playlist abgespielten Mediums werden angezeigt. Ein Drehen des Master-Reglers navigiert innerhalb des Playlist–Ansicht. In der Ansicht *Playlists* gibt es zusätzlich die Funktion *(Exit)*, die zum Hauptmenü führt.

5.3.3 Clock

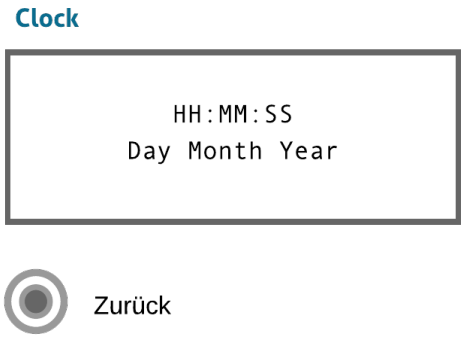

Abbildung 5.5: Ansicht des Menüpunkts *Clock*

Der Menüpunkt *Clock* (siehe Abb. 5.5) zeigt die aktuelle Uhrzeit, sowie das Datum an. Ein Drücken des Master-Reglers führt zurück zum Hauptmenü.

5.3.4 Statistics

Abbildung 5.6: Ansicht des Menüpunkts *Statistics*

Der Menüpunkt *Statistics* (siehe Abb. 5.6) zeigt an, wieviele Interpreten, Alben und Songs aktuell in der Musikdatenbank insgesamt zu finden sind. Zusätzlich wird die Gesamtspieldauer angezeigt.

Ein Drücken des Master-Reglers führt zurück zum Hauptmenü.

5.3.5 Options

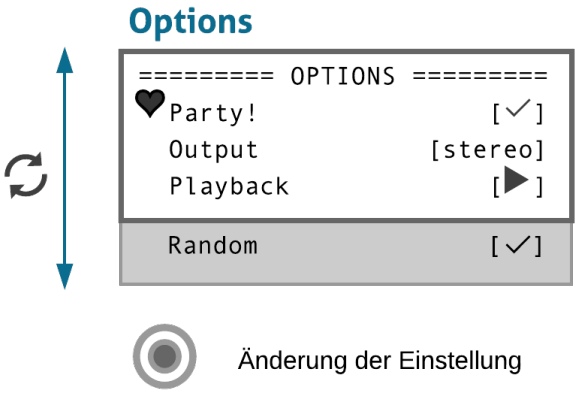

Abbildung 5.7: Ansicht der Kategorie *Options*

Options ist eine Kategorie, die keine Unteransichten hat (siehe Abb. 5.7). Optionen werden direkt durch ein Drücken des Master-Reglers geändert.

- **Party!:** LEDs können aktiviert oder deaktiviert werden (Aus/Ein).
- **Output:** Umschalten zwischen stereo- und mono-Ausgabe (stereo, mono).
- **Playback:** Wechsel zwischen Abspiel-, Pause-, und Stopmodus (Play, Pause, Stop).
- **Random:** Zufällige Wahl eines Musikstücks aus der aktuell gewählten Playlist (Aus/Ein).

5.3.6 Powermenu

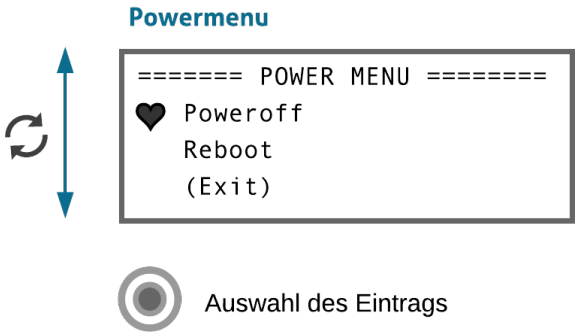

Abbildung 5.8: Ansicht des Menüpunkts *Powermenu*

Der Menüpunkt *Powermenu* (siehe Abb. 5.8) enthält die Funktionen *Poweroff* für das Herunterfahren des Systems und *Reboot* für einen Neustart des Systems. Mit *(Exit)* gelangt man zurück ins Hauptmenü.

5.3.7 Systeminfo

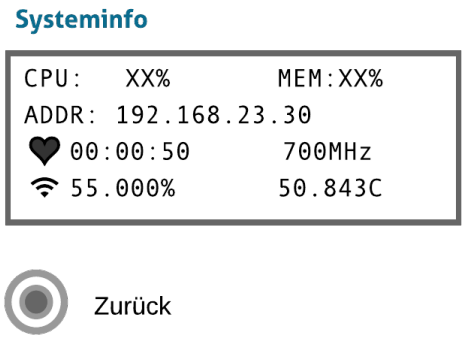

Abbildung 5.9: Ansicht des Menüpunkts *Systeminfo*

Der Menüpunkt *Systeminfo* (siehe Abb. 5.9) zeigt folgende Informationen zum System an:

- CPU-Auslastung in Prozent.
- Speicher-Auslastung in Prozent.
- IP-Adresse, Systemlaufzeit,
- WLAN-Empfangsstärke in Prozent.
- CPU-Temperatur in Celsius.

Ein Drücken des Master-Reglers führt zurück zum Hauptmenü.

5.3.8 About

Abbildung 5.10: Ansicht des Menüpunkts *About*

Der Menüpunkt *About* (siehe Abb. 5.10) zeigt Informationen zu den Entwicklern des Internetradios an.

5.4 Shortcuts

Ein Direktzugriff auf das *Powermenu* (siehe 6.2.7) ist durch Drücken des Master-Reglers von drei Sekunden möglich — in allen Anzeigemodi. Das führt zu mehr Komfortabilität in der Bedienung. Es kann direkt ins *Powermenu* gewechselt werden, ohne die aktuelle Anzeige verlassen zu müssen, um manuell zum *Powermenu* zu navigieren.

In der Ansicht *Now Playing* führt ein längeres Drücken (mindestens 600ms) des Master-Reglers ins Hauptmenü. Eine Abweichung zu der Funktionsweise in den anderen Ansichten war notwendig, weil auch der Wechsel zwischen Pause/Play möglich sein sollte. Für eine möglichst intuitive Bedienung bot es sich an, dies durch ein einfaches Drücken des Master-Reglers zu realisieren. Für den Wechsel ins Hauptmenü wurde deshalb ein längerer Drück–Zeitraum gewählt.

5.5 Fazit

Die Möglichkeiten für die Darstellung der Inhalte waren aufgrund des 4x20 Displays verständlicherweise begrenzt. Auch die Bedienung des Menüs über einen einzigen Drehimpulsgeber setzt klare Grenzen für die Steuerung.

Trotz dieser minimalistischen Umstände, ist es gelungen, eine weitgehend konsistente Menüführung umzusetzen. Nach einer kurzen Eingewöhnungsphase sollte das Bedienkonzept gut funktionieren.

6 Zusammenfassung

Das selbstgesetzte Ziel — mit möglichst geringem finanziellen Einsatz (ca. 23 €) ein Internetradio auf Basis eines *Raspberry Pi* zu entwickeln — kann durchaus als erfolgreich betrachtet werden.

Verbesserungsmöglichkeiten zum Projekt wurden bereits in den jeweiligen Kapiteln aufgezeigt. Obwohl es online einige Internetradio–Projekte gibt, so ist *Eulenfunk* aufgrund der selbst gesetzten Anforderungen durchaus als Unikat zu bezeichnen und unterscheidet sich so grundlegend von den besagten Projekten, welche meistens einen jeweils ähnlichen Ansatz verfolgen.

Die Dokumentation wurde absichtlich in der Art einer *Bauanleitung* verfasst, da es sich hierbei um ein Free Software (und Hardware) Projekt handelt. Wir würden uns daher sehr freuen, wenn Andere *Eulenfunk* nachbauen und nach dem Open–Source–Gedanken weitere Verbesserungen und Vorschläge zurückfließen lassen.

Abschließend möchten wir sagen, dass wir durch das Projekt — gerade auf Hardwareebene — einiges gelernt haben und trotz des sehr knappen Zeitrahmens von ca. drei Wochen viel Spaß bei der Entwicklung des Internetradios hatten.

Literaturverzeichnis

[1] G. Wood and S. O'Keefe, "On Techniques for Content-Based Visual Annotation to Aid Intra-Track Music Navigation." in *ISMIR*, 2005, pp. 58–65.

[2] W. Gay, *Raspberry Pi Hardware Reference*. Apress, 2014.

[3] M. Richardson and S. Wallace, *Make: Getting Started with Raspberry Pi*. Maker Media, 2014.

[4] W. Gay, *Mastering the Raspberry Pi*. Apress, 2014.

[5] W. Gay, *Experimenting with Raspberry Pi*. Apress, 2014.

[6] B. Horan, *Practical Raspberry Pi*. Apress, 2013.

[7] A. R. & Mike Cook, *Projekte mit dem Raspberry Pi:*. mitp/bhv, 2014.

[8] D. Molloy, in *Exploring Raspberry Pi*, John Wiley & Sons, Inc., 2016.

[9] R. Suehle and T. Callaway, *Hacks für Raspberry Pi*. O'Reilly Verlag, 2014.

[10] S. Pietraszak, *Das Buch zu Raspberry Pi mit Linux*. O'Reilly Verlag, 2014.

[11] T. Warner, *Hacking Raspberry Pi*. Pearson Education, 2013.

[12] M. Schmidt, *Raspberry Pi: Einstieg - Optimierung - Projekte*. Dpunkt.Verlag GmbH, 2014.

[13] S. Sabarinath, R. Shyam, C. Aneesh, R. Gandhiraj, and K. P. Soman, "Accelerated FFT Computation for GNU Radio Using GPU of Raspberry Pi," in *Computational intelligence in data mining - volume 2: Proceedings of the international conference on cidm, 20-21 december 2014*, C. L. Jain, S. H. Behera, K. J. Mandal, and P. D. Mohapatra, Eds. New Delhi: Springer India, 2015.

[14] B. Smith, "A quick guide to GPLV3," *Free Software Foundation, Inc. Online: http://www. gnu. org/licenses/quick-guide-gplv3. html. Referred*, vol. 4, p. 2008, 2007.

[15] R. Pike, "The Go Programming Language," *Talk given at Google's Tech Talks*, 2009.

[16] P. Seuntiens, I. Vogels, and A. Van Keersop, "Visual experience of 3D-TV with pixelated ambilight," *Proceedings of PRESENCE*, vol. 2007, 2007.

[17] W. Taymans, S. Baker, A. Wingo, R. S. Bultje, and S. Kost, "GStreamer application development manual (1.2. 3)," *Publicado en la Web*, 2013.

[18] I. Z. W. Group and others, "Zero configuration networking (Zeroconf)," *Accessed*, vol. 2, p. 27, 2013.