

# LuminoGlass



Physical Computing Projekt

Lena Rybak  
Maximilian Müller  
Nels Appold

Wintersemester 2016/2017  
Ostbayerische Technische Hochschule Amberg-Weiden

# Inhaltsverzeichnis

<b>1</b>	<b>Idee</b>	<b>3</b>
<b>2</b>	<b>Konzept und Anforderungen</b>	<b>4</b>
2.1	Hardware . . . . .	4
2.2	Gehäuse . . . . .	4
2.3	Spiele / Soziale Interaktion . . . . .	4
2.4	Selbst erweiterndes Funknetz . . . . .	5
2.5	Musiksteuerung . . . . .	5
2.6	Datenverarbeitung am Mikrocontroller . . . . .	5
2.7	Controller und GUI . . . . .	6
<b>3</b>	<b>Umsetzung</b>	<b>7</b>
3.1	Elektronik . . . . .	7
3.1.1	FLORA . . . . .	8
3.1.2	Aktorik . . . . .	9
3.1.2.1	nRF24L01+ . . . . .	9
3.1.2.2	NeoPixel-Ring . . . . .	10
3.1.3	Sensorik . . . . .	12
3.1.3.1	LSM9DS0 . . . . .	12
3.2	Controller . . . . .	12
3.2.1	Entwurfsmuster . . . . .	14
3.2.2	Anbindung Transmitter . . . . .	14
3.2.3	Abstraktion von Gläsern und Adressen . . . . .	14
3.2.4	Benutzeroberfläche . . . . .	15
3.2.5	Plugins . . . . .	15
3.2.5.1	Farben . . . . .	16
3.2.5.2	Animationen . . . . .	16
3.2.5.3	Timer . . . . .	17
3.2.5.4	Musik . . . . .	18
3.2.6	Musiksteuerung und Audioanalyse . . . . .	18
3.2.6.1	Musikbeispiele . . . . .	19
3.2.6.2	Analysealgorithmus . . . . .	22
3.2.6.3	Einsatz . . . . .	24
3.3	Funk . . . . .	25
3.3.1	Grundsätze . . . . .	25
3.3.2	Adressierung . . . . .	25
3.3.3	Datenpakete . . . . .	26
3.3.4	Reichweite . . . . .	26
3.3.5	Automatische Erweiterung . . . . .	28
3.4	Gestenanalyse . . . . .	29
3.5	Gehäuse und Design . . . . .	31

3.6 Mikrocontroller Software . . . . .	36
3.6.1 Glas . . . . .	36
3.6.2 Transmitter . . . . .	41
<b>4 Ausblick</b>	<b>42</b>
<b>Literaturverzeichnis</b>	<b>43</b>
<b>Anhang</b>	<b>44</b>

# 1 | Idee

Damit sich Menschen in ihrer Umgebung wohl fühlen, müssen verschiedene Faktoren gegeben sein. Einer davon ist die Beleuchtung. Wir werden morgens wach und abends müde; das Licht steuert den natürlichen Bio-Rhythmus unseres Körpers. Um diesen zu beeinflussen und nach unseren Wünschen zu verändern, versuchen wir mit dem Einsatz verschiedener Lichter und Spektren eine ideale Umgebungssituation zu schaffen.

Oft wird dieser Faktor vernachlässigt. Insbesondere viele Gastronomen nutzen das Potential einer ansprechenden Beleuchtung nicht aus.

Hier setzt unsere Idee der dezentralen Beleuchtung an. Wir möchten mit dem LUMINOGLASS klassische, fest installierte Lampen mit kleinen, smarten und Mobile Leuchtelementen ergänzen. Um diese den Menschen besonders nahe zu bringen und den Nutzen zu maximieren, integrieren wir sie im ersten Schritt direkt in seinen unmittelbaren persönlichen Raum - sein Getränk. Das Konzept an sich ermöglicht darüber hinaus eine Integration der Leuchtelemente in nahezu jeden Gegenstand und an jeden Ort.

Durch die Mobilität der Lichter, passt sich die Beleuchtung der Gruppendynamik an. Soziale Interaktion wird durch integrierte Spiele unterstützt.

Jedes LUMINOGLASS ist individuell konfigurierbar und das Gesamtsystem beliebig erweiterbar. Das System ist dadurch nicht nur auf gewerbliche Nutzung begrenzt, sondern kann auch in kleiner Stückzahl privat eingesetzt werden.

Durch eingebaute Sensorik erkennt das LUMINOGLASS Bewegungen, kombiniert sie zu Gesten und kann darauf angemessen reagieren. Der Trinkende wird nicht vom Licht gestört.

Umfangreiche Controller Software mit eingebauter live-Musikerkennung ermöglicht eine präzise, schnelle und einfache Konfiguration.

# 2 | Konzept und Anforderungen

## 2.1 Hardware

Der Platz unter einem Glas ist begrenzt, weshalb als mobile Steuerungseinheit nur ein kleiner Mikrocontroller in Frage kommt. Aufgrund der passenden Form, der Energieeffizienz und der standardisierten Programmierung empfiehlt sich hier die Nutzung eines Arduino kompatiblen ARM Mikrocontrollers. Um größtmögliche Flexibilität bei der Beleuchtung zu garantieren, müssen angeschlossene LEDs einzeln ansteuerbar sein und einen möglichst großen Farbraum abdecken. Die verwendete Sensorik zur Bewegungserkennung muss, genau so wie die verwendeten Funkmodule, neben einer hohen Genauigkeit und Geschwindigkeit, kleine Abmessungen und eine hohe Energieeffizienz aufweisen.

## 2.2 Gehäuse

Für die Hardware jedes einzelnen LUMINOGLASS wird ein Gehäuse benötigt, welches am Boden eines Cocktailglases befestigt ist. Darin ist die Elektronik versteckt. Der Unterbau soll 3D modelliert und anschließend mit einem 3D Drucker ausgedruckt werden.

## 2.3 Spiele / Soziale Interaktion

Mithilfe des LUMINOGLASS soll die Interaktion zwischen den Gästen gesteigert werden. Folgend sind ein paar Ideen für Spiele aufgelistet, die diese zwischen den trinkenden Gästen fördern und unterstützen soll.

**Countdown** Für besondere Anlässe wie Silvester oder das ‚Reinfeiern‘ in einen Geburtstag, ist ein Countdown ideal. Mithilfe von sekündlichem Blinken des LUMINOGLASS in einer beliebigen Farbe kann ein Countdown rhythmisch unterstützt werden.

Eine beliebige Zeit (z.B. 10, 20 oder 30 Sekunden) kurz vor Mitternacht kann der Timer gestartet werden. So unterstützt das LUMINOGLASS den gemeinschaftlichen Sprechchor in den letzten 10 Sekunden vor dem Jahreswechsel oder Geburtstag. Ist der Timer abgelaufen, so kann eine beliebige Aktion des Glases gewählt werden, wie zum Beispiel ein schnelles Blinken des Lichts.

**Getränk especial** Bietet die Location in regelmäßigen Zeitabständen besondere Aktionen, kann ein Timer gesetzt werden, der den Gästen signalisiert, dass es nun Zeit ist das Special wahrzunehmen. Die Signalisierung kann durch verschiedene Effekte (Blinken, Pulsieren, etc.) in einer Signalfarbe erfolgen. Nach einer bestimmten Zeit kann wieder in den Ausgangszustand gewechselt werden.

**Zufallsspecial** Alle 30 Minuten blinkt für eine gewisse Zeit ein zufälliges Glas. Schafft der Besitzer dieses Glases es innerhalb dieser Zeit mit einem leeren Glas an die Bar, bekommt er z.B. Rabatt auf sein nächstes Getränk.

**Anstoßen** Es werden unterschiedlich viele Startgläser gewählt. Jedes dieser Gläser erhält eine individuelle Farbe. Durch das Anstoßen dieser Startgläser mit anderen Gläsern wird die entsprechende Farbe verbreitet. Die Farbe, in welcher die meisten Gläser am Ende leuchten gewinnt das Spiel und kann durch z.B. ein Getränkespecial belohnt werden.

## 2.4 Selbst erweiterndes Funknetz

Das Gesamtkonzept des LUMINOGLASS stellt eine dezentrale Lösung in den Mittelpunkt. Somit muss eine Datenübertragung über Funk ermöglicht werden, da sich eine kabelgebundene Lösung durch ihre geringe Flexibilität disqualifiziert. Trotz des Umstandes, dass die einzelnen Netzwerkteilnehmer sich nicht in direkter Nähe zum Server befinden und aufgrund des Akkubetriebs die zur Verfügung stehende Sendeleistung begrenzt ist, muss jeder Netzwerkteilnehmer erreicht werden können. Hierbei bietet es sich an auf die Lösung eines selbst erweiternden Funknetzes zu setzen.

Die Idee hinter diesem Netz ist, dass jeder Empfänger auch gleichzeitig als Sender agiert. So können Clients über andere Clients hinweg erreicht werden und das Reichweiteproblem gelöst.

## 2.5 Musiksteuerung

Die Wirkung von Musik kann mit rhythmischen Lichtwechseln verstärkt werden. Dieser Effekt wird durch die Nähe der Gläser zum Besitzer verstärkt. Bisherige Lösungen sind in der Regel stationär und mit sehr hellen Leuchtmitteln ausgestattet, was als störend empfunden werden kann. Eine dezentrale, mobile Lösung mit vielen kleinen Leuchtmitteln kann hier Abhilfe schaffen oder ein bestehendes System ergänzen. Um das System möglichst einfach zu halten, ist neben einer manuellen Steuerung auch ein Automatikmodus vorgesehen.

## 2.6 Datenverarbeitung am Mikrocontroller

Innerhalb der Datenverarbeitung des Mikrocontrollers gibt es verschiedene Aufgabenbereiche:

1. Ansteuerung der LEDs
2. Datenpakete empfangen und auswerten
3. Datenpakete senden
4. Gyroskop auswerten

Um LEDs anzusteuern und hierbei Funktionalitäten wie das Blinken mit einer festen Frequenz oder der Möglichkeit eines Farbverlaufs umzusetzen, werden feste Grundfrequenzen bei der Verarbeitung benötigt. Um diese festen Grundfrequenzen zu realisieren, bietet sich die Nutzung der internen Hardware-Timer des Mikrocontroller an, um Interrupts mit einem festen Zeitabstand auszuführen.

Datenpakete werden asynchron versendet. Der Mikrocontroller weiß nicht, wann ein Datenpaket von einem anderen Netzwerkteilnehmer gesendet wird. Um dennoch alle Pakete

zu empfangen, muss möglichst oft zyklisch abgefragt werden, ob ein neues Paket vorhanden ist.

Datenpakete müssen nur gesendet werden, wenn Anfragen vorhanden sind, bzw. empfange Pakete weitergeleitet werden sollen. Die Datenverarbeitung kann hier also synchron ablaufen.

Ebenso wie die Paketverarbeitung muss das Gyroskop regelmäßig ausgelesen werden, jedoch nicht mit einer festen Frequenz, da es sich nicht um einen zeitkritischen Vorgang handelt. Aus diesem Grund bietet es sich an, die Auswertung des Gyroskops ebenfalls, wie die Paketverarbeitung, in einer unterbrechbaren Schleife auszuführen.

Durch die Aufteilung der Aufgaben in verschiedene Zeitscheiben bzw. Zeitbereiche kann außerdem die Last des Mikrocontrollers pro Zyklus reduziert und die Bearbeitungsgeschwindigkeit erhöht werden.

## 2.7 Controller und GUI

Der Controller und die GUI sind das zentrale Steuerelement des Systems. Hier ist, neben den geforderten Funktionen, eine einfache Bedienbarkeit, eine einheitliche Benutzerführung und eine ansprechende Optik besonders wichtig. Für einzelne Funktionalitäten im Bereich der Musiksteuerung ist die Möglichkeit der Tastatursteuerung vorgesehen. Folgende Funktionen sollen im Controller vorhanden sein:

- Steuern aller Farb- und Beleuchtungsvariationen eines Glases
- Animationen bzw. Farbverläufe mit beliebig vielen Farben
- Zeitgesteuerte Aktionen (Wecker, Timer, Countdown)
- Musiksteuerung mit Automatikmodus und Tastatursteuerung

Alle Funktionen sollen als *Plugins* ausgelegt sein und eine einfache Erweiterung unterstützen. Gewisse zentrale Funktionalitäten wie die Adressierung und das Zurücksetzen der Gläser sind im Hauptprogramm integriert.

# 3 | Umsetzung

## 3.1 Elektronik

Die Elektronik des Glases setzt sich aus einem Adafruit FLORA v3, einem nRF24L01+ Funkmodul, einem NeoPixel-Ring, einem LSM9DS0 und einem LiPo-Akku zusammen.

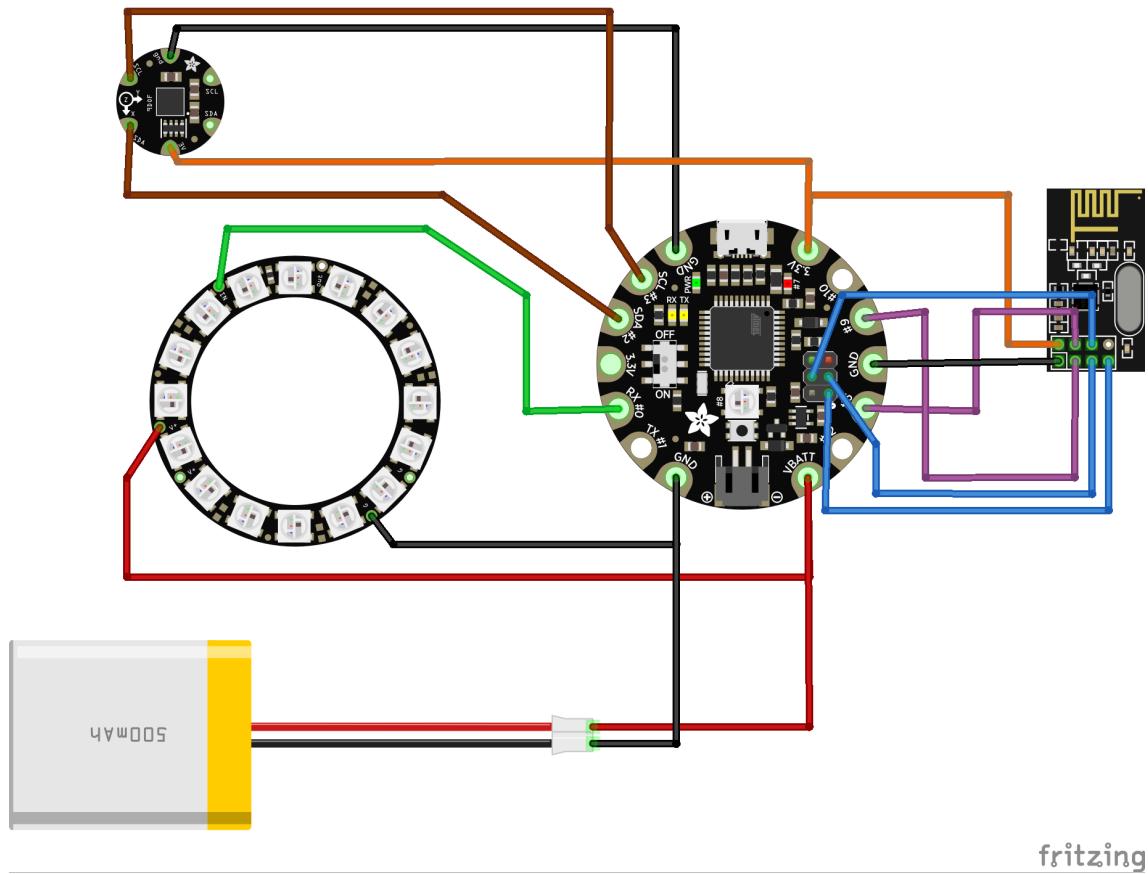
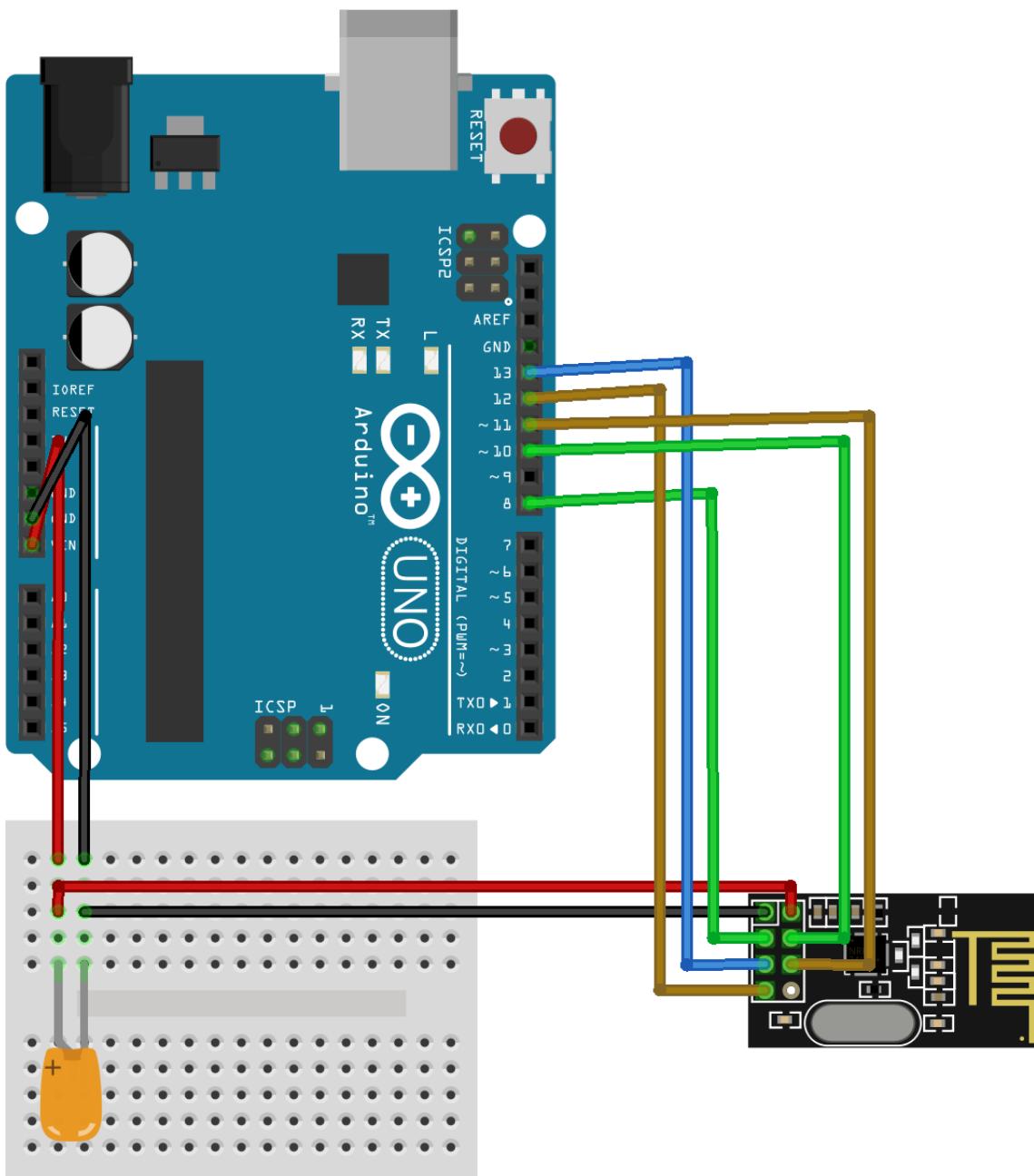


Abbildung 3.1: Übersicht Verschaltung der Elektronik des Glas

Neben der Elektronik des Glases, ist auch die Elektronik des Transmitters relevant. Der Transmitter setzt sich dabei aus einem Arduino v3, einem Kondensator zur Spannungsstabilisierung und einem nRF24L01+ Funkmodul zusammen.



fritzing

Abbildung 3.2: Übersicht Verschaltung der Elektronik des Transmitters

### 3.1.1 FLORA

Der Adafruit FLORA v3 ist ein Mikrocontroller mit einer runden Form. Er basiert auf dem Arduino Konzept der Open-Hardware und verwendet einen ATMEGA32u4 Prozessor mit einer Taktfrequenz von 8MHz als Chip. Des Weiteren besitzt er diverse In- und Outputs, welche auf den äußereren Ring geführt wurden.

Der Mikrocontroller kann, wie in Abbildung 3.1 dargestellt, mittels Mirco-Usb oder einem LiPo-Akku über eine JST-Schnittstelle mit Strom versorgt werden.

Des Weiteren ist eine NeoPixel LED auf dem Board integriert, wodurch die Möglichkeit einer Zustandsanzeige geschaffen und das Debugging vereinfacht wird.

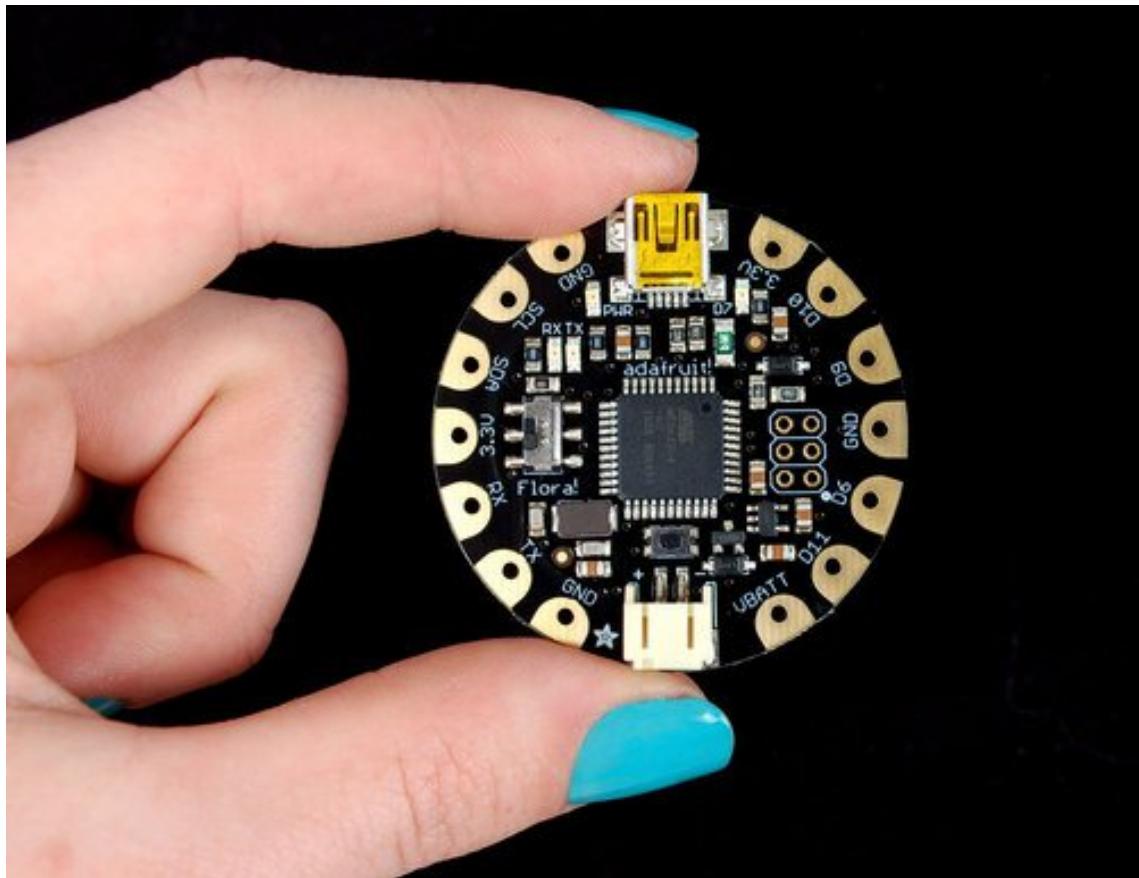


Abbildung 3.3: Adafruit FLORA v3<sup>1</sup>

### 3.1.2 Aktorik

#### 3.1.2.1 nRF24L01+

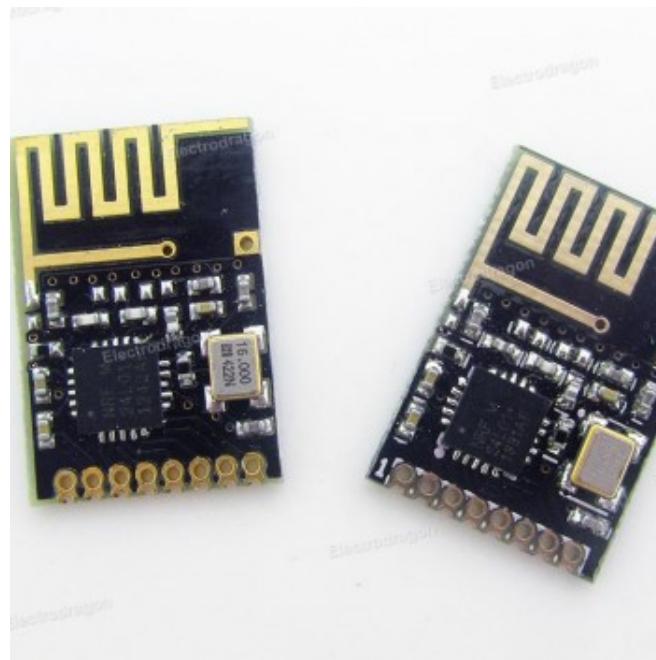


Abbildung 3.4: Mini nRF24L01+<sup>2</sup>

Das nRF24L01+ Funkmodul ist ein Transceiver-Chip, welcher im 2,4 GHz Frequenzband arbeitet. Er zeichnet sich durch geringe Kosten und einen geringen Stromverbrauch aus. Des Weiteren kann er durch die Verwendung einer Funkfrequenz im 2.4GHz Band frei eingesetzt werden, da die Nutzung dieses Frequenzbandes nicht gesetzlich reglementiert ist.

Durch den Anschluss des Funkmoduls an die SPI-Schnittstelle entsteht keine Konkurrenz mit anderen Sensoren und Aktoren bezüglich der vorhandenen Anschlüsse des FLORAs. Mit einer Größe von lediglich  $18 \times 12.5 \times 2\text{mm}$  in der Mini-Ausführung, passt sich das Modul in das auf geringe Größe ausgelegte Konzept optimal ein. Zudem ist keine weitere Antenne notwendig.

### 3.1.2.2 NeoPixel-Ring

Bei dem NeoPixel-Ring handelt es sich um eine Weiterentwicklung der NeoPixel-Module der Firma Adafruit. Diese Module bestehen aus einzelnen RGBW-LEDs, welche durch eine eigene Schaltung angesteuert werden können. Bei dem NeoPixel-Ring handelt es sich um eine ringförmige Verschaltung dieser NeoPixel-Module.

Der gesamte Ring wird hierbei über eine einzige Datenleitung angesteuert. Es kann jedes NeoPixel-Modul auf dem Ring unabhängig angesteuert und geschaltet werden. Außerdem besitzt der verwendete NeoPixel-Ring den gleichen Durchmesser wie der FLORA, was eine Anordnung übereinander ermöglicht, ohne weiteren Platz in der Breite zu beanspruchen. Der Ring kann durch Spannungen zwischen 3V und 7V betrieben werden, was ihn unabhängig von der Spannungsregulierung des Mikrocontrollers macht. Er ist direkt an die Stromversorgung angeschlossen werden. Aufgrund der 16 sehr hellen LEDs, bietet der Ring genug Leuchtkraft für die Beleuchtung eines Glases.

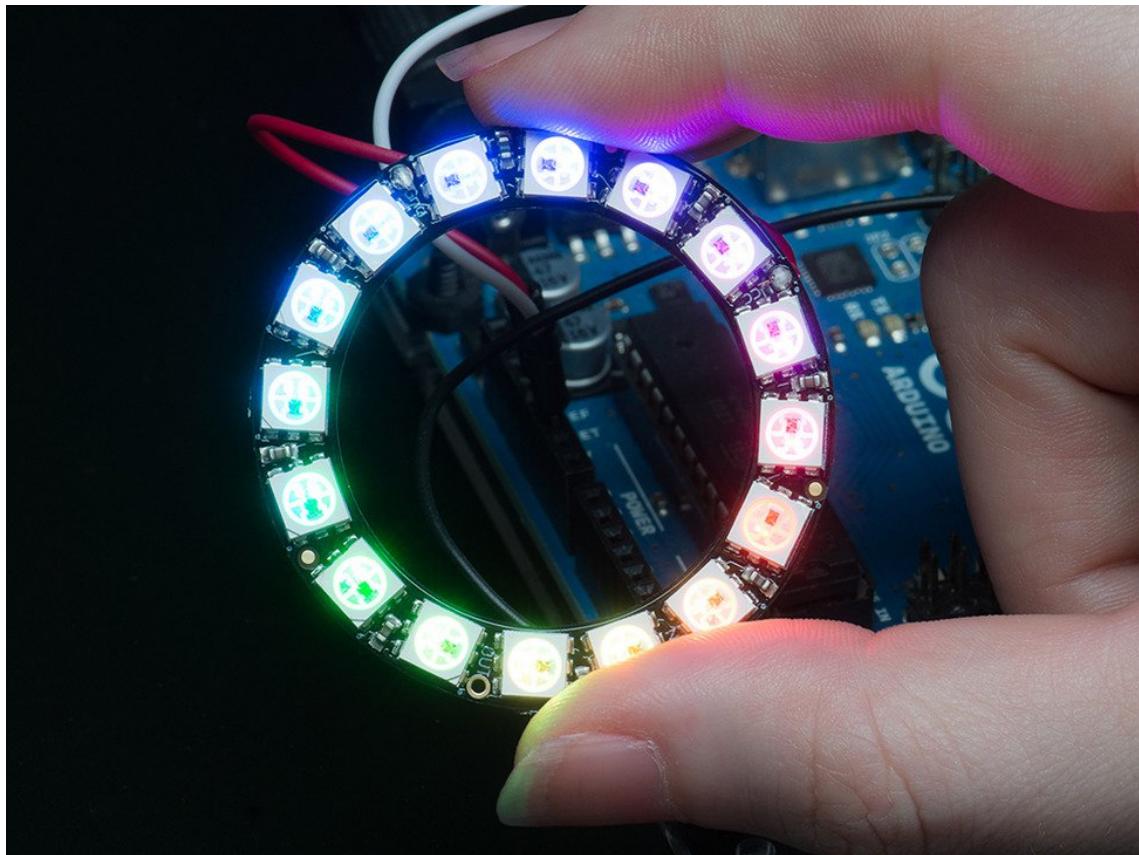


Abbildung 3.5: NeoPixel Ring<sup>3</sup>



Abbildung 3.6: Beleuchtetes Lumino Glass

### 3.1.3 Sensorik

#### 3.1.3.1 LSM9DS0

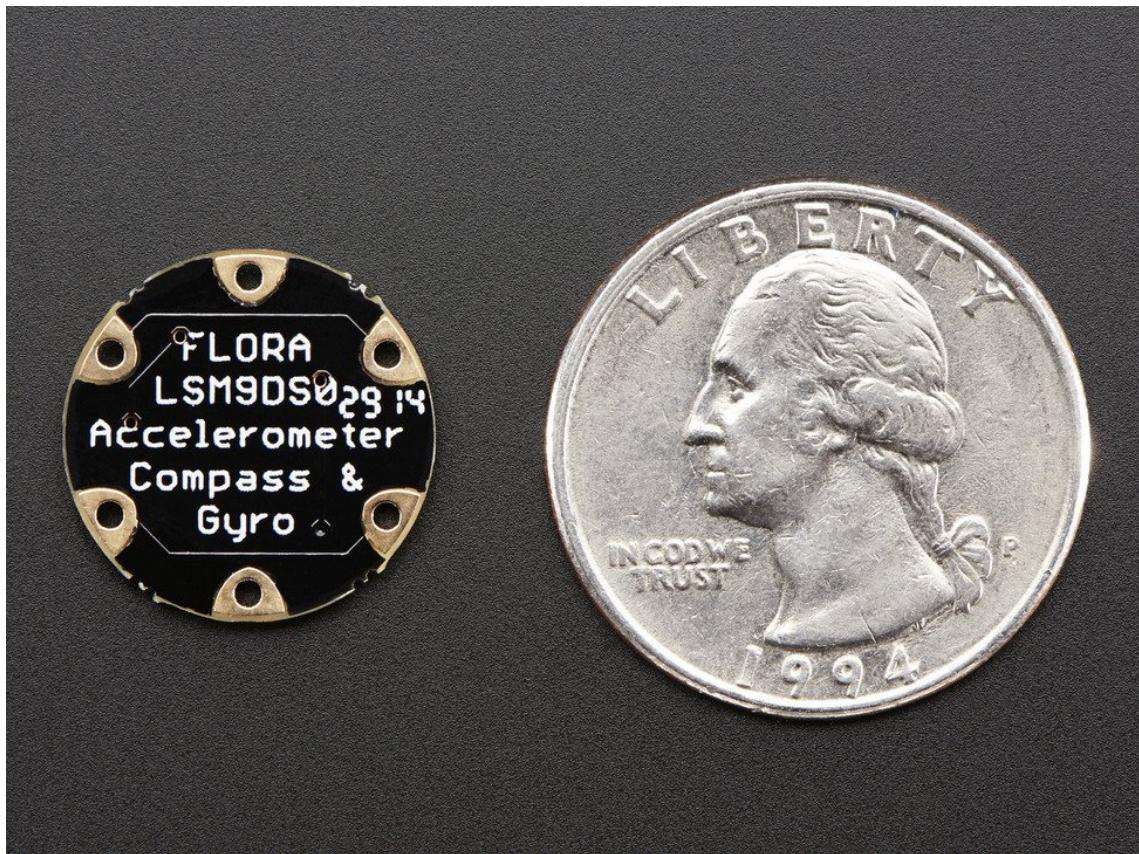


Abbildung 3.7: LSM9DS0<sup>4</sup>

Das LSM9DS0-Modul der Firma Adafruit ist ein 9-DOF<sup>5</sup> Accelerometer, Gyroskope, Magnetometer und Thermometer. Es wird mittels  $I2C$ <sup>6</sup> angesprochen. Hierdurch können bei gleichzeitiger Einsparung von Anschlüssen, alle Funktionen schnell und einfach angesteuert und alle Daten ausgelesen werden.

Auch das LSM9DS0-Modul ist sehr klein gehalten und kann, aufgrund seiner Runden Form, im Inneren des NeoPixel-Rings platziert werden.

## 3.2 Controller

Um alle Gläser im Netzwerk korrekt zu steuern und deren Nutzung zu ermöglichen, steht eine umfangreiche Controller-Software zur Verfügung. Sie übermittelt mithilfe des Transmitters (Abschnitt 3.6.2 auf Seite 41) sämtliche Steuerungsdaten. Die Software ist in C# entwickelt und nutzt das WPF<sup>7</sup> Framework für die Benutzeroberfläche. Um ein zeitgemäßes Design zu ermöglichen, ist das *Metro* Design Kit von *Mahapps* eingebaut. Zur Nutzung des Controllers ist minimal das .Net Framework Version 4.6 erforderlich.

<sup>5</sup>Dimensions of Freedom

<sup>6</sup>Inter-Integrated Circuit Bus

<sup>7</sup>Windows Presentation Foundation

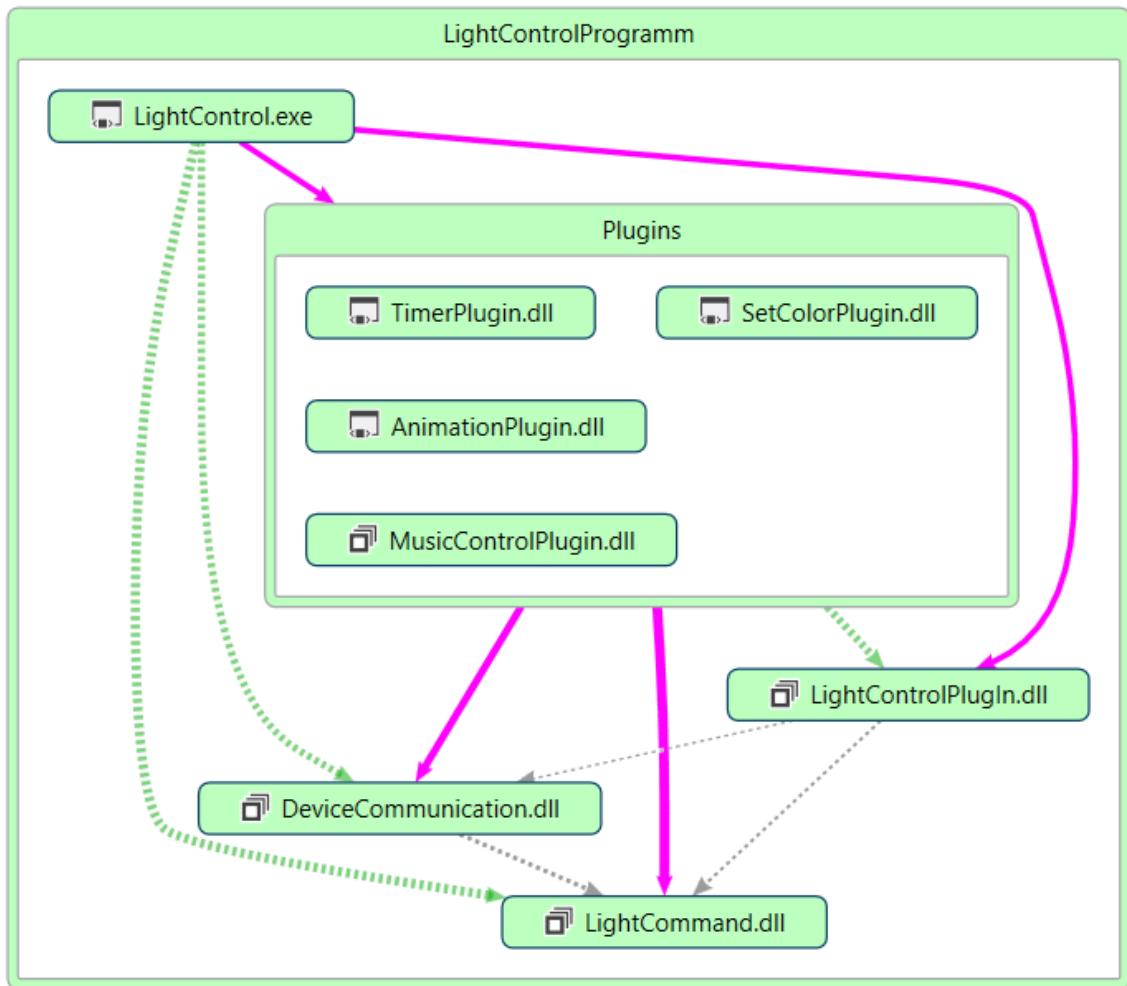


Abbildung 3.8: Code Map

Wie in Abbildung 3.8 zu sehen, ist der Controller grundsätzlich in fünf Teile aufgeteilt.

1. Die LightControl.exe ist die Hauptanwendung selbst.
2. Die Plugins sind Erweiterungen zum Hauptprogramm, die die eigentliche Anwendelogik enthalten
  - (a) Das SetColorPlugin zum Einstellen von Farben und Effekten
  - (b) Das TimerPlugin für zeitgesteuerte Aktionen der Gläser
  - (c) Das AnimatinPlugin für automatische Farbverläufe
  - (d) Das MusicControlPlugin zur Musiksteuerung
3. Das LightControlPlugin stellt die Schnittstelle zwischen Plugins und der LightControl.exe zur Verfügung
4. Die DeviceCommunication stellt die Schnittstelle zum Funknetzwerk bereit
5. Die LightCommand enthält das Funkprotokoll

Durch diese Trennung ist es möglich, die Anwendung sehr einfach um weitere Plugins zu erweitern oder Teile zu verändern, ohne die Gesamtanwendung neu zu Kompilieren. Weiterhin wird die Kopplung vermindert und der Zusammenhalt erhöht, da zusammengehörige Softwarekomponenten eine feste Einheit bilden.

### 3.2.1 Entwurfsmuster

Um den Quelltext übersichtlich zu halten wurden folgende Entwurfsmuster eingesetzt:

- Model-View-ViewModel
- Property Injection
- Plugin

Um die Oberfläche responsive zu halten und ein Einfrieren zu verhindern, sind alle zeitlich aufwändigen Datenbindungen über das ViewModel ans Modell über `async/await` und Tasks realisiert.

### 3.2.2 Anbindung Transmitter

Der Transmitter ist mit der maximalen seriellen Datenübertragungsrate von 57600 Baud/s angebunden. Über den Zugriff auf Windows-Verwaltungsobjekte ist es möglich, die Serielle-Verbindung automatisch auf dem richtigen Com Port herzustellen. Der Benutzer muss hier nicht eingreifen. Um das System möglichst stabil zu halten, sendet der Controller mit einer maximalen Rate von 25 Paketen pro Sekunde. Werden mehrere Pakete gleichzeitig gesendet, steht ein Zwischenpuffer mit 8 Speicherplätzen zur Verfügung. Dieser löscht bei Überfüllung automatisch die ältesten Pakete.

Um den Zugriff auf die Geräte zur Netzwerkkommunikation zu vereinfachen und zu vereinheitlichen steht die `IDeviceCommunication` Schnittstelle im `DeviceCommunication` Namespace bereit. Siehe Quelltext Ausschnitt 3.1

```
1 public interface IDeviceCommunication
2 {
3     event CommunicationEventHandler DataReceived; //Event Handler for
4     received Data
5     bool ImportantOnly { get; set; } //Set Communication Device to Important
6     Communication Only
7     void Send(ILightCommand command, bool isImportant = false); //SendData
8     Packet
9     bool OpenConnection();
10    bool CloseConnection();
11 }
```

Quelltext Ausschnitt 3.1: Controller - `IDeviceCommunication`

### 3.2.3 Abstraktion von Gläsern und Adressen

Im Controller werden alle Kommunikationsteilnehmer grundsätzlich als `IDeviceDescription`, siehe Quelltext Ausschnitt 3.2 auf der nächsten Seite, gehandhabt. Für Gläser besteht die Implementierung der Schnittstelle als `GlassDeviceDescription`. Wie auch das reale Glas enthält sie sowohl die native GUID, als auch die zugewiesene Adresse. Weiterhin wird über die Schnittstelle ein direktes Senden und Empfangen von `Commands` an bzw. von diesem Gerät ermöglicht.

```

1 public interface IDeviceDescription
2 {
3     event DeviceDescriotionEventHandler DataAvailable;
4     bool LockDevice { get; set; }
5     long DeviceAddress { get; set; }
6     long NativeAddress { get; set; }
7     DeviceCommunication.IDeviceCommunication CommunicationInterface { get;
8         set; }
9     LightCommand.ILightCommand LastCommand { get; set; }
10    void SendData(LightCommand.ILightCommand command, bool isImportant =
11        false);
12}

```

Quelltext Ausschnitt 3.2: Controller - IDeviceDescription

**Adressierung und Such von Netzwerkteilnehmern** Wie in Abschnitt 3.3.1 auf Seite 25 beschrieben, ist das Funknetz nicht darauf ausgelegt, verlässlich Daten auszutauschen. Um trotzdem eine möglichst störungsfreie und korrekte Suche nach Teilnehmern des Netzwerks zu ermöglichen, fordert die Suche exklusiven Zugriff auf den Transmitter und sendet alle Kommandos mehrfach. Jegliche Parallele Kommunikation wird währenddessen unterbunden. Das Protokoll sieht folgende Schritte zur Suche vor:

1. Warte 500ms, bis die gesamte Kommunikation im Netz zum Erliegen gekommen ist
2. Broadcast Ping Paket
3. Warte 2s
4. Füge empfangene Absenderadressen zu Liste hinzu, falls diese dort noch nicht vorhanden waren
5. Wiederhole Schritte 2-4 mehrfach, um den Einfluss von Störungen zu minimieren
6. Weise jeder empfangenen nativen Adresse eine Adresse des Standard Adressraums zu
7. Benachrichtige jedes Glas einzeln zwei mal mit einem Ping Command, welches die neue Standard Adresse enthält

### 3.2.4 Benutzeroberfläche

Die Hauptbenutzeroberfläche stellt die nötigen Funktionalitäten für das Laden, die Verwaltung und die Umgebung für die in ihr enthaltene Plugins (siehe Abschnitt 3.2.5) bereit. Dazu gehört eine Implementierung der *IDeviceCommunication* Schnittstelle (siehe Abschnitt 3.2.2 auf der vorherigen Seite), eine Implementierung des Protokolls und der Adressierung (siehe Abschnitt 3.2.3) sowie das Vorgeben grundsätzlicher Ressourcen wie Styles oder Icons.

### 3.2.5 Plugins

Plugins stellen im Controller die eigentliche Anwenderfunktionalität bereit. Der Kern jedes Plugins ist die Implementierung der *ILightControlPlugIn* Schnittstelle aus dem *LightCon-*

trolPlugIn Namensraum, siehe Quelltext Ausschnitt 3.3.

```

1 public interface ILightControlPlugIn
2 {
3     PlugInState State { get; set; } //State (active, inactive)
4     IDeviceCommunication Com { set; } //Communication to Wireless Network
5     UserControl View { get; } //View for Main Application
6     IHeaderGenerator HeaderGenerator { get; set; } //Header Generator for
7         Addressing
8     string Name { get; } //Plugin Name
9     void Exit(); //Exit Method, called when Main Application closes
}

```

Quelltext Ausschnitt 3.3: Controller - ILightControlPlugIn

### 3.2.5.1 Farben

Das Farben Plugin befindet sich in der *SetColorPlugin.dll* und stellt eine Oberfläche zur Konfiguration aller enthaltenen Farben und Animationseffekte eines Glases bereit. Über Presets können bestimmte Kombinationen aus Effekten und Farben, sehr komfortabel mit einem Klick, geladen werden. Die Farbauswahl selbst erfolgt über das *Color Wheel* Benutzersteuerelement von *Andrew Syrov*.

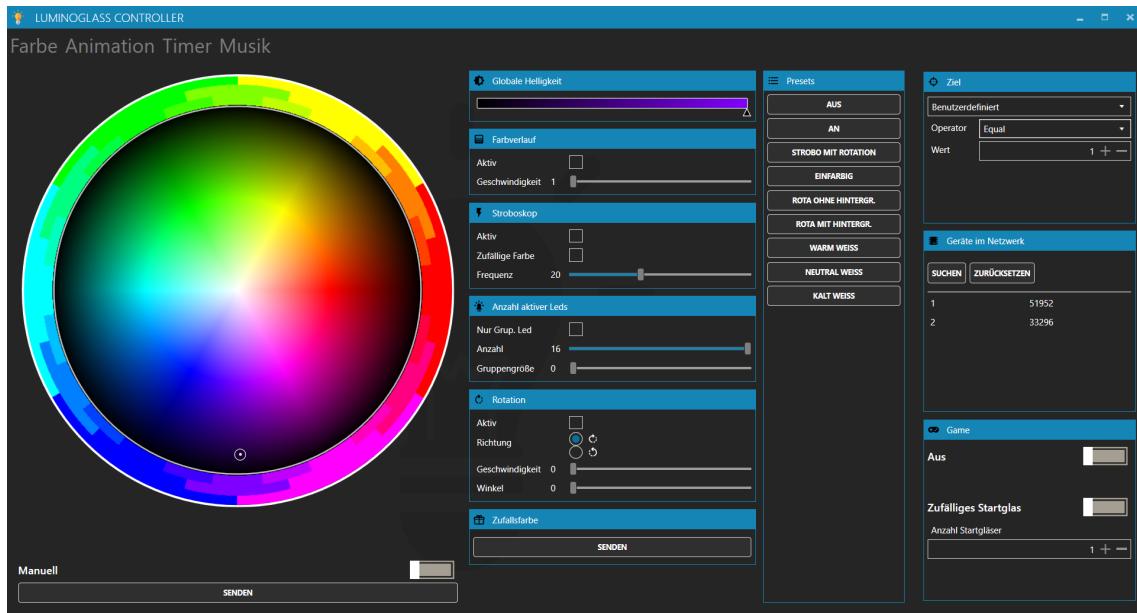


Abbildung 3.9: Plugin Farbe

### 3.2.5.2 Animationen

Das Animationen Plugin befindet sich in der *AnimationPlugin.dll* und stellt eine Oberfläche für Farbverläufe bereit. Hier können über einstellbare Farbmodi, oder benutzerdefiniert, aufeinander abgestimmte Farbpaletten erzeugt und als Animationen mit einstellbarer Farbverlaufsdauer gestartet werden. Die Anzahl der im Farbverlauf enthaltenen Farben lässt sich frei bestimmen.

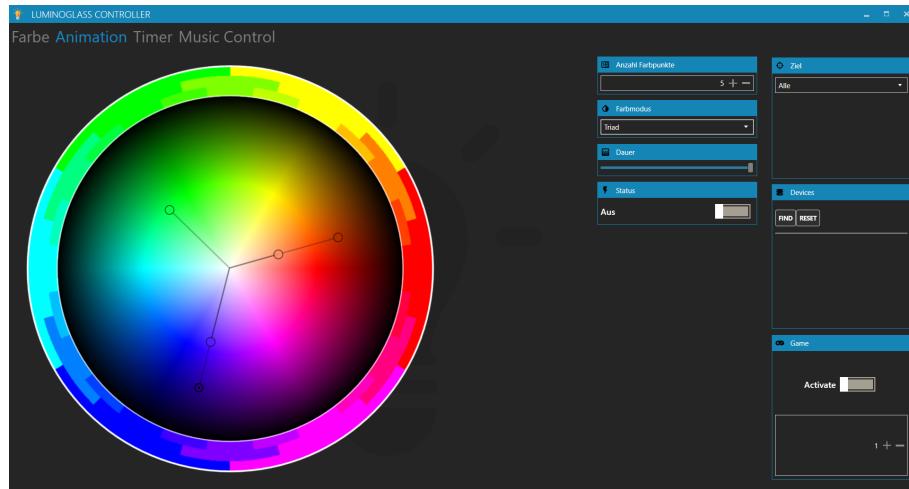


Abbildung 3.10: Plugin Animation

### 3.2.5.3 Timer

Das Timer Plugin befindet sich in der *TimerPlugin.dll* und stellt eine Oberfläche für zeitgesteuerte Aktionen bereit. Hier stehen drei verschiedene Zeitauslöser je zwei mal zur Verfügung, welche verschiedene vorgegebene Effekte mit einstellbaren Farben auslösen.

- Alarm: Ähnlich einem Wecker können hier Effekte zu einem festen Zeitpunkt ausgelöst werden.
- Timer: Ähnlich einer Eieruhr läuft die eingestellte Zeit beim starten ab und löst bei Ablauf einen Effekt aus. Timer können sich automatisch neu starten.
- Countdown: Stellt eine Kombination aus Alarm und Timer dar. Hier kann eine feste Uhrzeit und eine Anzahl an *Countdown-Sekunden* eingestellt werden. *Countdown-Sekunden* vor der eingestellten Uhrzeit wird sekündlich der Countdown Effekt ausgelöst, zur eingestellten Uhrzeit der Alarm Effekt.

Folgende Effekte stehen zur Verfügung und können (bis auf Einschalten und Ausschalten) mit verschiedenen Farben ausgelöst werden:

1. Einschalten
2. Ausschalten
3. Kurzes Pulsieren
4. Langes Pulsieren
5. Kurzes Stroboskop
6. Langes Stroboskop
7. Farbwechsel

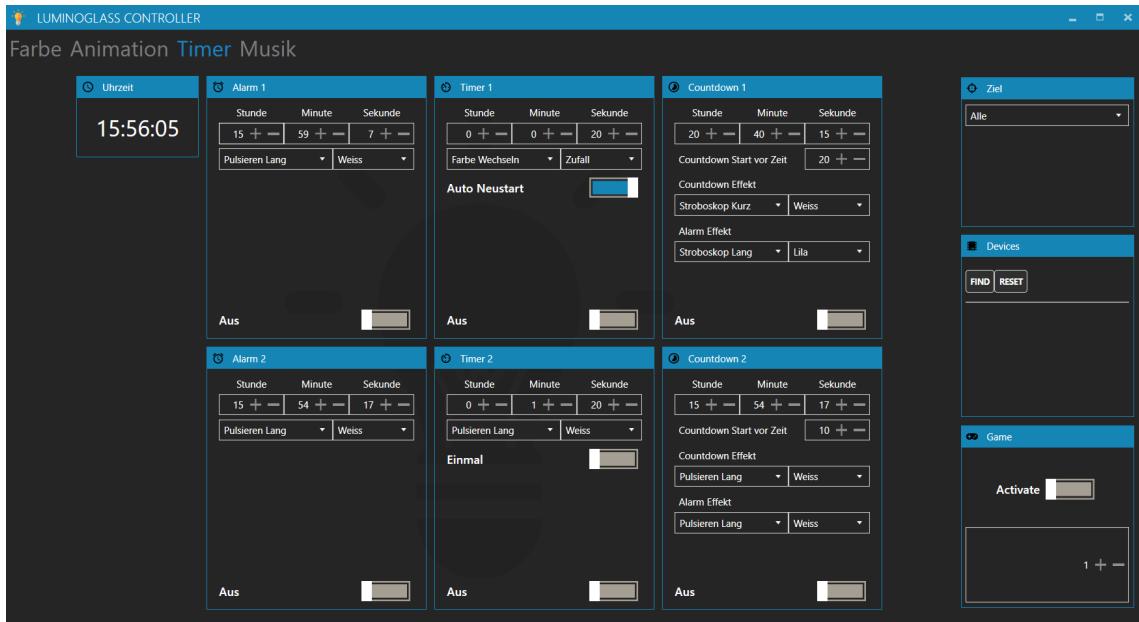


Abbildung 3.11: Plugin Timer

### 3.2.5.4 Musik

Das Musik Plugin befindet sich in der *MusicControl.dll* und stellt eine Oberfläche und Algorithmen zur Musiksteuerung bereit. Siehe Abschnitt 3.2.6. Die Spektralgrafik wird mit OxyPlot gezeichnet.

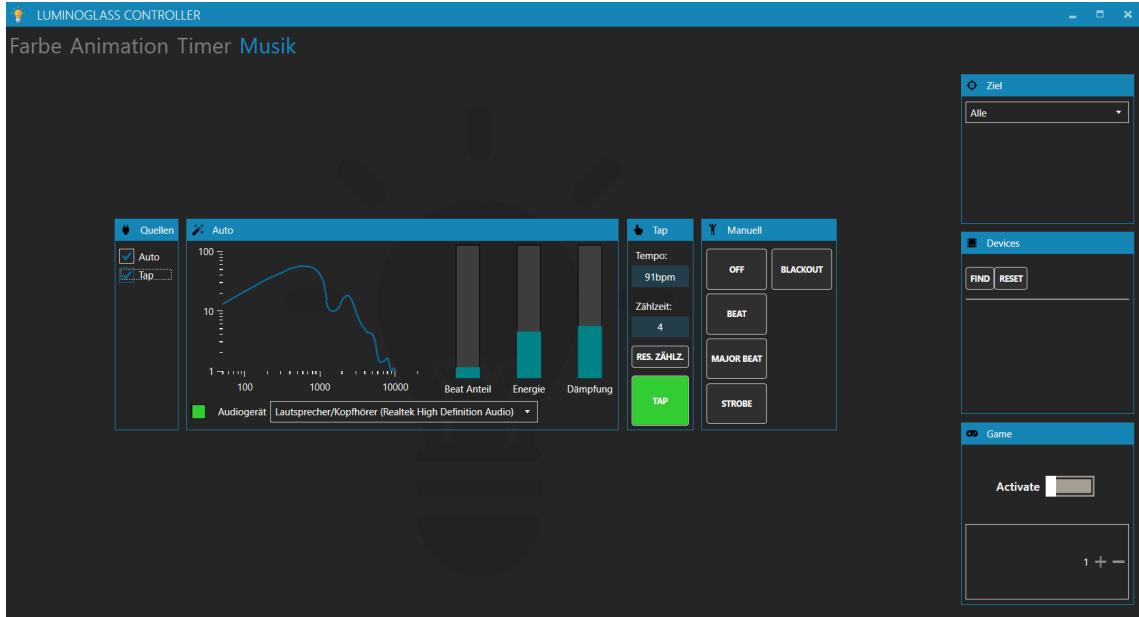


Abbildung 3.12: Plugin Musik

### 3.2.6 Musiksteuerung und Audioanalyse

Das Konzept des LUMINOGLASS ist unter Anderem für Discotheken und Clubs entwickelt und soll dort durch dezentrale Beleuchtung die Lichtstimmung verbessern. Ein zentraler Aspekt dabei ist die Steuerbarkeit zur Musik. In der Regel ist die aktuell gespielte Musik sehr perkussiv und basslastig. Um die Gläser passend dazu anzusteuern, liegt es nahe, die

Beats oder auch Amplitudenpeaks im Bassbereich (ca. 40Hz-200Hz) auszuwerten und als Trigger für Farb- oder Effektwechsel des Glases zu verwenden.

Oft wird diese Audioauswertung rein über Amplitudenpeaks in einem gewissen Frequenzbereich durchgeführt. Dies hat in unserem Test allerdings zu sehr ungenauen und unbefriedigenden Resultaten geführt. Weiterhin ist es für eine solche Auswertung notwendig manuell einen Amplitudenwert festzulegen, ab dem ein Peak als Beat erkannt wird.

Die Audioaufzeichnung und die Fourier Transformation werden von der NAudio Bibliothek übernommen.

### 3.2.6.1 Musikbeispiele

Da sich verschiedene Musikgenres sehr stark unterscheiden und die Musiksteuerung möglichst flexibel sein soll, ist eine Spektralanalyse von Audiobeispielen aus den relevanten Genres notwendig.

**Pop / Charts** Aktuell generell sehr populäre Musikstücke sind im relevanten Frequenzbereich oft sehr einfach und durchgängig. Ein konkretes Beispiel hierfür ist *David Guetta, Would I Lie To You*. Auf der Spektralanalyse (Abbildung 3.13) sind deutlich die durchgängigen Beats erkennbar. Ein weiteres Beispiel ist *Rag'n'Bone Man, Human* (Abbildung 3.14 auf der nächsten Seite). Hier sind einzelne Abschnitte im Bass Bereich deutlich voneinander getrennt. Sie stellen zwar keine Beats im eigentlichen Sinne dar, eignen sich im Block allerdings gut als Trigger für einen Lichtwechsel.

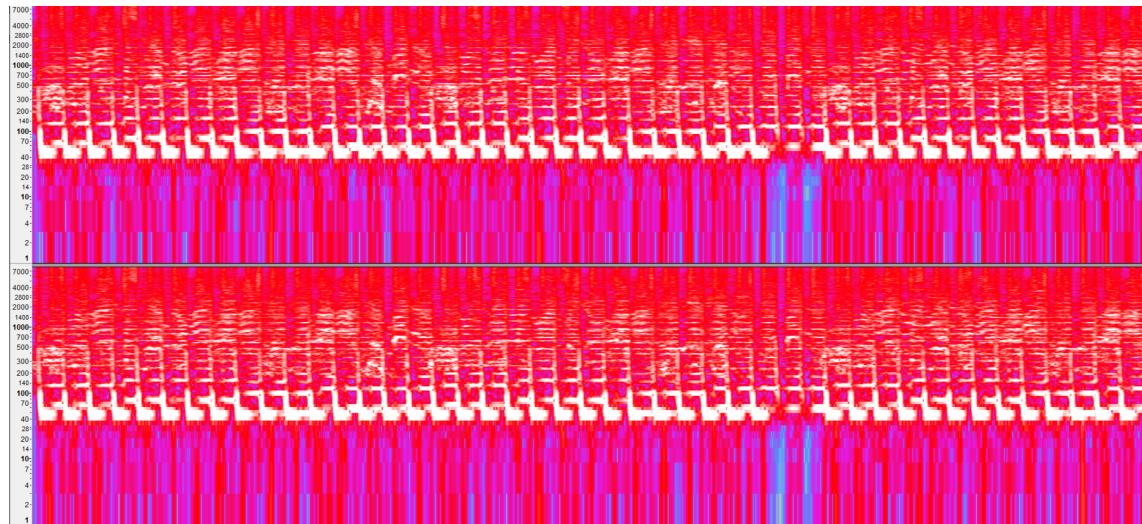


Abbildung 3.13: David Guetta, Would I Lie To You

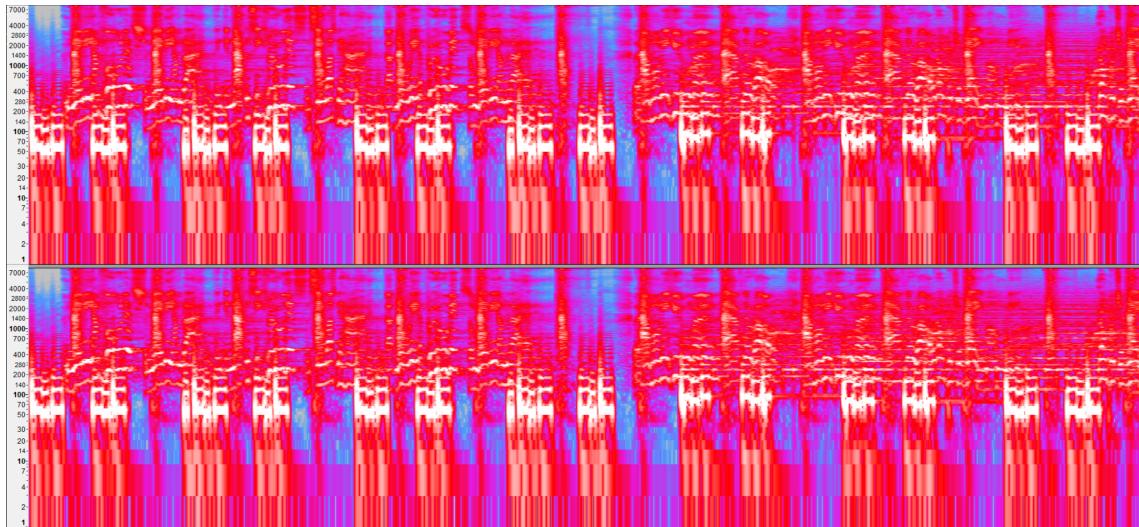


Abbildung 3.14: Rag'n'Bone Man, Human

**Elektro** Bei elektronischer Musik sind sehr häufig durchgehende Basslines zu sehen. Zwei gute Beispiele hierfür sind *Skrillex, Scary Monsters and nice Sprites*(Abbildung 3.15) und *Seven Lions, She Was*(Abbildung 3.16 auf der nächsten Seite). Hier sind durchgängige horizontale Linien im Spektrum zu sehen, die sich mit Beats überlagern. Eine reine Beatkennung auf Amplitudenbasis würde hier nur eingeschränkt funktionieren.

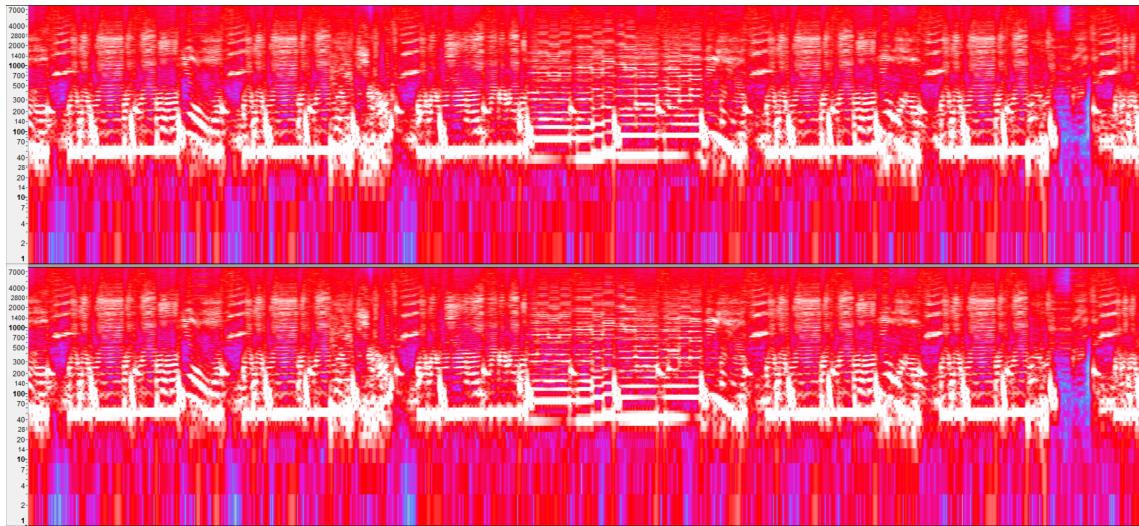


Abbildung 3.15: Skrillex, Scary Monsters And Nice Sprites

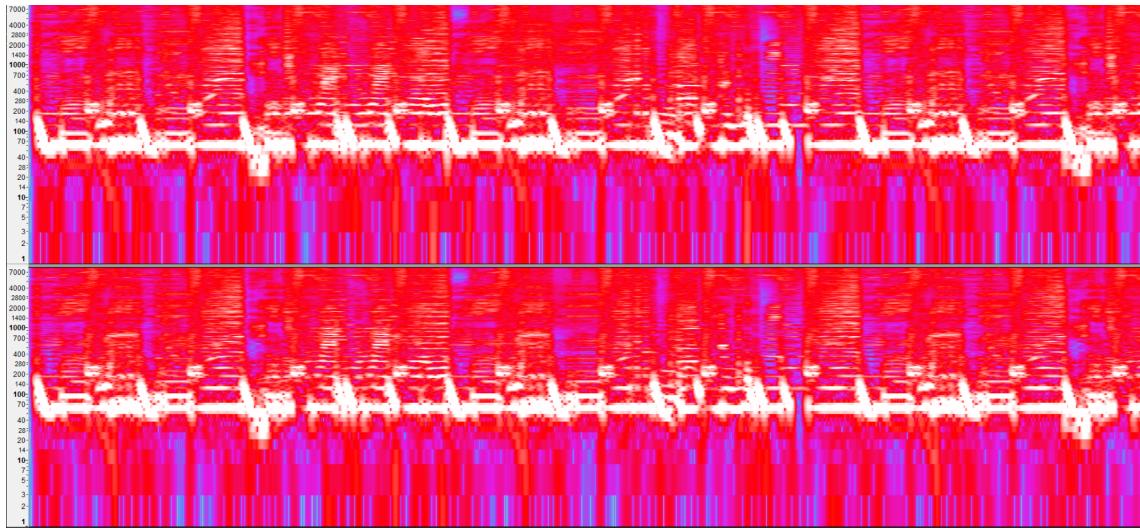


Abbildung 3.16: Seven Lions, She Was

**Rock und Metal** Rock und Metal ist im Gegensatz zu den bereits behandelten Genren vorwiegend akustische Musik. Hier kann eine automatische Beaterkennung sehr schwierig werden, da die Qualität der Aufnahme und Nachbearbeitung des Musikstücks eine große Rolle spielt. Hierbei ist es wichtig, dass die Frequenzbereiche der Instrumente voneinander getrennt sind und die Beats nicht zu schnell aufeinander folgen. Dieser Kontrast wird in den nachfolgenden Spektren deutlich. Obwohl die Beats bei *Parkway Drive*, *Wild Eyes*(Abbildung 3.18 auf der nächsten Seite) für Menschen gut zu hören sind, erinnert das Spektrum durch die Überlagerung der Instrumente und der hohen Dichte an Peaks an ein Durcheinander. Im Gegensatz dazu stellt *Slash, Bent To Fly*(Abbildung 3.17) ein recht sauberes Spektrum dar.

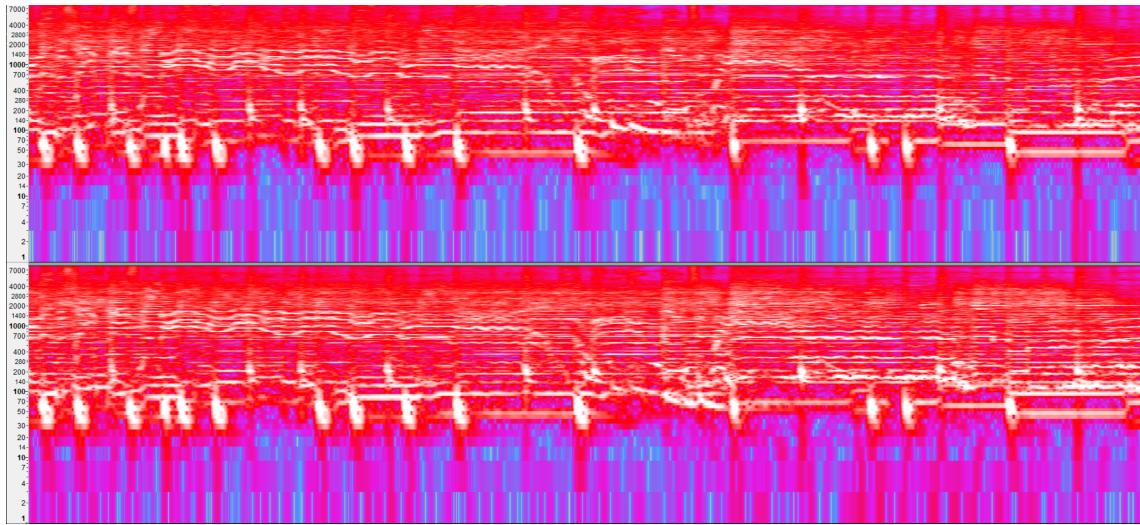


Abbildung 3.17: Slash, Bent to Fly

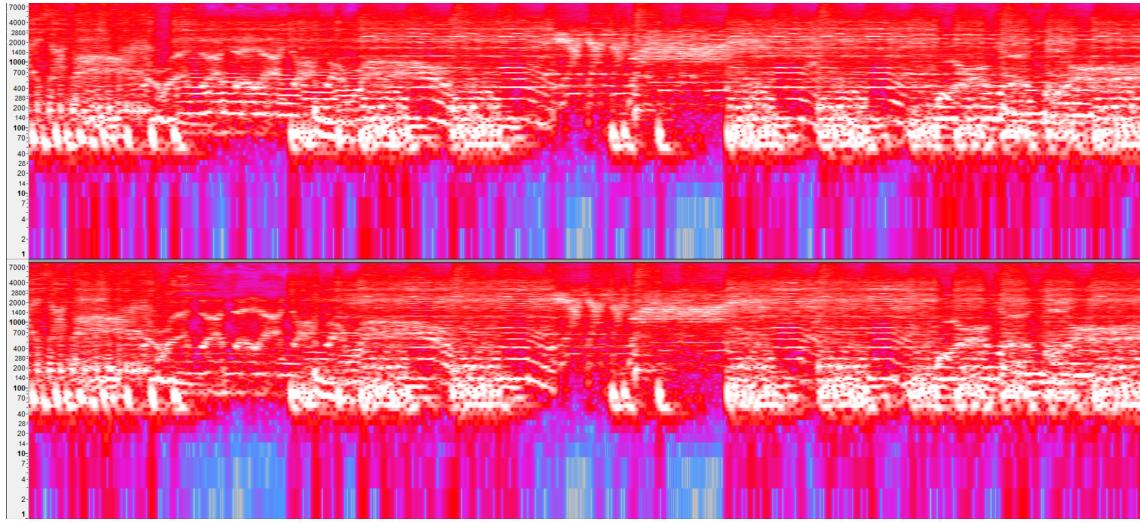


Abbildung 3.18: Parkway Drive, Wild Eyes

### 3.2.6.2 Analysealgorithmus

Um die Erkennung von Beats möglichst sauber und genau durchzuführen basiert der Algorithmus auf folgenden Schritten:

1. Aufzeichnung von 1024 Samples (Mono Signal)
2. Fast Fourier Transformation mit Länge 1024. Daraus ergibt sich Spektrum zum Zeitpunkt  $n$  mit 512 Werten, da FFT Achsen symmetrisch zum Mittelpunkt  $\Rightarrow X(k, n)$
3. Umrechnung in das Frequenzspektrum aus  $X(k, n) \Rightarrow X(f, n)$
4. Berechnung der Energie des Beatspektrums mit  $0Hz \leq f_{beatMax} < 240Hz$

$$E_{beat}(n) = \sum_{f=0}^{f_{beatMax}} X(f, n)$$

5. Berechnung der Gesamtenergie des Signals mit  $0Hz \leq f_{max} < \frac{f_a}{2}$   $f_a$ : Abtastfrequenz

$$E_{total}(n) = \sum_{f=0}^{f=f_{max}} X(f, n)$$

6. Gesamtenergie in gleitenden Mittelwertfilter einfügen

$$E_{totalMean}(n) = \frac{1}{m} \cdot \sum_{i=0}^{m-1} E_{total}(t - i) \text{ mit } m + 1 = 256$$

7. Berechnung des Faktors zwischen den letzten beiden Beatenergien

$$BeatFakt(n) = \frac{E_{beat}(n)}{E_{beat}(n - 1)}$$

8. Berechnung des Beat-Anteils des Spektrums

$$BeatShare(n) = \frac{100 \cdot E_{beat}(n)}{E_{total}(n)}$$

9. Beat-Anteil in gleitenden Mittelwertfilter einfügen. Ergebnis des Filters

$$BeatShare_{mean}(n) = \frac{1}{m} \cdot \sum_{i=0}^{m-1} Beatshare(n-i) \text{ mit } m+1 = 256$$

10. Reduzieren des Spektrums  $X(f, n)$  auf den, für die Weiterverarbeitung relevanten Bereich  $X_r(f, n)$  unter  $f_{beatMax}$
11. Bestimmen des Maximums der Ableitung

$$Dev_p(n) = \max \left\{ \frac{d}{dn} X_r(f, n) \right\}$$

12. Maximum der Ableitung in gleitenden Maximumfilter einfügen

$$Dev_{max}(n) = \max \{ Dev_p(n-0), \dots, Dev_p(n-m) \} \text{ mit } m+1 = 48$$

13. Vergleichsfaktor für Ableitungsänderung aus Dämpfung  $D$  berechnen

$$Dev_{comp}(n) = Dev_{max}(n) \cdot \log_{40} \frac{D(n)}{1.8} \text{ mit } 10 \leq D(n) \leq 40$$

### Bedingungen für einen Beat

1.  $Dev_p(n) > Dev_{comp}(n) \equiv$  Aktuell maximale Amplitude größer dem berechneten Vergleichswert
2.  $Dev_p(n) > 0.8 \equiv$  Aktuell maximale Amplitude größer als Fix Wert (Rauschunterdrückung)
3.  $E_{beat}(n) > D(n) \equiv$  Aktuelle Beat-Energie größer Dämpfung
4.  $BeatFakt(n) > 1 \equiv$  Aktuelles Spektrum enthält mehr Energie im Beat Bereich als Letztes
5.  $E_{total}(n) > 1.1 \cdot E_{totalMean}(n) \equiv$  Aktuelles Spektrum enthält mehr als 10% mehr Energie als Durchschnitt
6.  $E_{totalMean}(n) > 4 \equiv$  Durchschnittsenergie größer 4. Erfahrungswert, der ungewolltes Auslösen in Liedern mit starker Dynamik verhindert.
7. Wenn kein Anderer Beat ‚aktiv‘ ist wird ein Beat auf dieser Frequenz ‚aktiviert‘ und ein Lichtwechsel ausgelöst. Falls bereits ein Beat ‚aktiv‘ war wird die Dämpfung um Faktor 0.2 erhöht
8. Um einen ‚aktiven‘ Beat zurückzusetzen, muss die Amplitude Spektrums an der Frequenz des Beats um Faktor 0.5 abgesunken sein, oder die Amplitude selbst einen sehr kleinen Wert besitzen ( $< 0.01$ )

Die konkreten Werte der Verhältnisse wurden empirisch ermittelt. Die gleitenden Mittelwertfilter mitteln über 256 Werte, was bei einer Analysefrequenz von  $\approx 50Hz$  etwa einem Zeitraum von  $12s$  entspricht. Der gleitende Maximumfilter läuft über 48 Werte, was einem Zeitraum von ca.  $2,4s$  entspricht. Bildlich gesprochen zieht der Algorithmus Linien über die Maxima des Spektrums im Bassbereich. Siehe Abbildung 3.19 auf der nächsten Seite

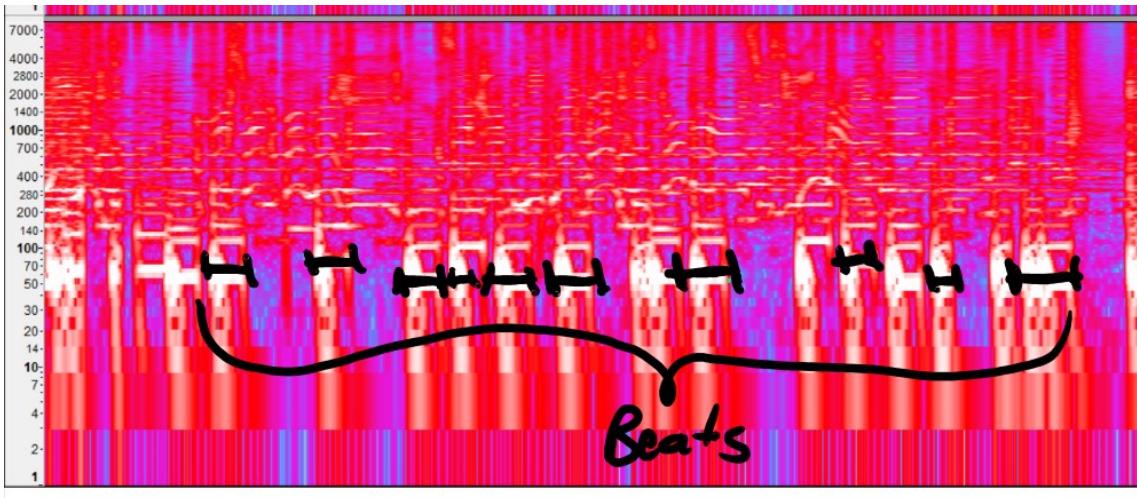


Abbildung 3.19: Bildliche Darstellung Algorithmus

**Dämpfung** Um anpassungsfähig zu sein, besitzt der Algorithmus einen Dämpfungsfaktor. Dieser befindet sich im Bereich  $10 \leq D \leq 40$ . Wird ein Beat erkannt, wird die Dämpfung um 0.5 erhöht; wird ein Beat bei aktiver Blockierung erkannt, wird die Dämpfung um 20% erhöht. Jedes mal, wenn  $BeatFakt(n) < 1$ , wird die Dämpfung um 8% reduziert.

**Automatischer Stroboskop Effekt** Wenn  $BeatShare > 4 \cdot BeatShare_{mean}$  wurde eine Passage mit sehr hohem Beat Anteil gefunden. Der Stroboskopeffekt wird aktiviert. Wenn  $BeatShare(n) < BeatShare_{mean}(n)$  oder  $E_{beat}(n) < 5$  ist die Passage zu Ende und der Stroboskopeffekt wird beendet.

**Major und Minor Beats** Tritt ein besonders starker Beat  $BeatFakt(n) > 0.1 \cdot D(n)$  (*Major Beat*) auf, so ändert sich neben der Lichtfarbe auch der Lichteffekt. Andernfalls ändert sich lediglich die Lichtfarbe (*Minor Beat*).

**Automatisches Ausschalten** Wenn  $E_{beat}(n) < 0.1$  wird das Licht ausgeschaltet.

### 3.2.6.3 Einsatz

Um einen praktikablen Einsatz der Musikerkennung zu ermöglichen, ist das gesamte Plugin über die Tastatur steuerbar, sogar wenn die Musikerkennung nicht explizit als aktives Plugin ausgewählt ist. Zusätzlich zur Automatik ist es möglich ein Tempo zu ‚Tappen‘ und mit der Zählzeit des Liedes zu synchronisieren. Da die Musikerkennung auf Beats und nicht auf Zählzeiten abgestimmt ist, ergänzen sich die beiden Methoden sehr gut. Sie können einzeln aktiviert bzw. deaktiviert werden.

Zusätzlich ist es möglich alle Funktionen und Effekte manuell auszulösen. Es stehen folgende Kommandos zur Verfügung:

- Stroboskop ‚S‘ (So lange Schaltfläche oder Taste gehalten wird)
- Major Beat ‚M‘ (Farb- und Effektwechsel)
- Minor Beat ‚Leertaste‘ (Farbwechsel)
- Blackout ‚B‘ (Licht aus, so lange Schaltfläche oder Taste gehalten wird)

## 3.3 Funk

Die Datenübertragung wird mit einem *nRF24L01+* Funkmodulen von *NordicSemic* realisiert. Das selbst erweiternde Funknetz wird mit dem im folgenden beschriebenen Protokoll aufgebaut und setzt auf der *RadioHead* Bibliothek von Mike McCauley auf. Sie implementiert die Datenübertragung und die Ansteuerung des Funkmoduls und stellt Methoden auf einer entsprechend hohen Ebene der Abstraktion zur Verfügung.

### 3.3.1 Grundsätze

Das entworfene Funkprotokoll ist verbindungslos und auf hohen Durchsatz von Broadcast Paketen optimiert. Es findet daher keine Prüfung der Integrität oder auf korrekten Empfang des Pakets statt. Das Risiko, dass Pakete in seltenen Fällen nicht ihr Ziel erreichen, wird hingenommen. Allerdings kann das Risiko durch mehrfaches Senden von besonders wichtigen Kommandos stark reduziert werden. Weiterhin kann eine Controller-Applikation exklusives Funkrecht einfordern und somit ein Überlagern von Kommandos verhindern.

### 3.3.2 Adressierung

Um die Sozialen Interaktion und eine individuelle Ansteuerung der Gläser zu ermöglichen, benötigt jedes Glas eine eigene Adresse. Diese sollte einzigartig und dem Controller bekannt sein.

Aus diesem Grund gibt es im Controller die Möglichkeit, die Teilnehmer des Netzwerks, welche sich in Reichweite zueinander befinden, zu suchen. Dies geschieht über einen *Ping*-Befehl, woraufhin ein Austausch von Adressen erfolgt. Im Anschluss besitzt jeder erreichte Teilnehmer eine eigene Adresse, welche dem Controller bekannt ist.

Über diese Adresse kann jedes Glas exklusive angesprochen werden, allerdings lassen sich mit dieser Adresse auch weitere Ansprechverhalten erzeugen. So ist es möglich die Adressen mittels eines Addressoperators einer weiteren Operation zu unterziehen. Folgende Operatoren stehen zur Verfügung:

- Gleich (Adresse wird auf Gleichheit geprüft)
- Ungleich (Adresse wird auf Ungleichheit geprüft)
- Modulo Gleich 0 (Adresse wird mit Rest durch den gesendeten Wert geteilt, bei Rest gleich 0 Annahme des Pakets)
- Modulo Ungleich 0 (Adresse wird mit Rest durch den gesendeten Wert geteilt, bei Rest ungleich 0 Annahme des Pakets)
- Größer (Adresse ist größer als gesendeter Wert)
- Kleiner (Adresse ist kleiner als gesendeter Wert)

Mittels dieser Operatoren wird eine Gruppierung von Adressen ermöglicht.

Damit der anfängliche Austausch der Adressen zwischen Gläsern und Controller möglich wird, benötigt jedes Glases zu Beginn eine möglichst eindeutige Id, die so genannte GUID. Diese wird zufällig erzeugt. Sie dient als native Adresse des Glases, solange keine explizite Adresse durch den Controller empfangen wurde.

Um eine möglichst große Anzahl von Gläsern anzusprechen und gleichzeitig den Datenverkehr und insbesondere die Datenmenge klein zu halten, wurden 2-Byte-Adressen eingefügt. Diese bieten einen theoretisch möglichen Adressraum von 65536 unterschiedlichen Adressen.

Dieser Adressraum ist in mehrere Bereiche aufgeteilt, um alle gewünschten Funktionalitäten zu bieten und gleichzeitig, bei geringem Mehraufwand in der Software, eine optimale Ausnutzung zu erreichen:

Adressbereich	Funktion
0x0	Controller
0x0001 – 0x7FFE	Explizite Adressen
0x7FFF – 0xFFFFD	Native Adressen
0xFFFFE	Default explizite Adresse
0xFFFFF	Broadcast

Tabelle 3.1: Aufteilung der 2-Byte-Adressen

### 3.3.3 Datenpakete

Für eine einfache Kommunikation zwischen den Gläsern und dem Controller ist jedes Datenpaket mit gleicher Länge und Aufbau definiert. Die Länge beträgt 16 Bytes. Hiervon gehören die ersten 7 Byte zum *Header* und die folgenden 9 Byte zum *Payload*. Durch diese geringe Paketgröße können schnell und häufig viele Pakete versendet werden und die Latenz bleibt entsprechend gering. Auch können weitere Kommandos ohne Probleme mit ins Protokoll aufgenommen werden. Die Aufteilung der Datenpakete ist in den folgenden Tabellen dargestellt: Abhängig vom *CommandType* werden unterschiedliche Daten in dem

Name	Size [Byte]
MessageId	1
SourceAddress	2
DestinationAddress	2
TimeToLive	1
AddressOperator	1

Tabelle 3.2: Aufbau Header eines Datenpakets

Name	Size [Byte]
CommandType	1
CommandData	8

Tabelle 3.3: Aufbau Payload eines Datenpakets

Bereich *CommandData* des *Payloads* gesendet. Dies ist in der folgenden Tabelle dargestellt:

### 3.3.4 Reichweite

Die Reichweite für die *nRF24L01+* Module wird mit bis zu 240m in offenen Bereichen angegeben (vgl. *NRF24 Mini SMD Wireless Module (NRF24L01 Compatible)*). Allerdings kann dies nur unter idealen Bedingungen erreicht werden.

Die reale Reichweite hängt von mehreren Faktoren ab:

1. Konfigurierte Ausgangsleistung
2. Konfigurierte Datenrate
3. Stabilität der Strom-/Spannungsversorgung

CmdNumber	Command	Parameter	Size [Byte]
0	Ping	NewAddress Generate GUID	2 1
1	Off		
2	On	Reset	1
3	Brightness	Value	1
4	GameMode	On/Off ActiveGame	1 1
		R G B W Fading	1 1 1 1 1
5	Color		
6	RandomColor	Fading	1
7	NumberActiveLED	Count GroupSize	1 1
		Direction Speed NumberLED Angle	1 1 1 1
8	Rotate		
9	Strobe	On/Off Frequency	1 1
		On/Off StandingTime FadingTime	1 1 1
10	FadeOut		
11	PingAnswer	GUID	2
		R G B W	1 1 1 1
12	ColorByClink		
		R G B W	1 1 1 1
13	Clink		

Tabelle 3.4: Derzeit verwendete Commands mit dem jeweiligen Payload

4. Zur Verfügung stehender Maximalstrom
5. Interferenzen mit anderen Funktechnologien auf gleicher Frequenz (z.B. Wifi, Bluetooth)
6. Sonstige Funk beeinflussende Umgebungsbedingungen (z.B. Dicke von Wänden, Metallische Abschirmungen, etc.)

Derzeit ist die Ausgangsleistung mit dem Maximum von 0dBm konfiguriert, des Weiteren ist die Datenrate mit dem Maximum von 2Mbps konfiguriert, was ebenfalls eine maximale Empfangsempfindlichkeit von -82dBm ermöglicht. Ein Problem stellt die Strom-Spannungsversorgung dar. Das Modul ist an den Spannungsregulator des Mikrocontrollers angeschlossen. Dieser stellt eine Spannung von 3,3V und einen Maximalstrom von 100mA bereit. Allerdings sind an diesen auch der Chip selber und die sonstige 3.3V Peripherie angeschlossen. Hierdurch kann es zu Spannungseinbrüchen kommen, was die Sende- und

Empfangsleistung verringert.

Des Weiteren ist der angedachte Haupteinsatzbereich des Glases innerhalb geschlossener Räume. In diesen sind heutzutage diverse Störungen durch Smartphones und ähnlichen vorhanden. Durch den Einsatzbereich verringert sich die effektive Reichweite drastisch. So haben wir, in einem Altbau mit zahlreichen aktiven WLAN Verbindungen, eine Maximalreichweiten von ca. 10 – 15m erreicht.

Diese doch recht geringe Reichweite wird durch das sich selbst erweiternde Funknetz teilweise kompensiert und auf eine dem Anwendungsbereich angemessene Größe erweitert.

### 3.3.5 Automatische Erweiterung

Um die Reichweite des Systems zu vergrößern und jedes Glas in einem Raum sicher zu erreichen, kann sich das Funknetz automatisch erweitern.

Diese Erweiterung wird dadurch erreicht, dass jedes Glas sowohl die Rolle des Empfängers als auch die des Senders einnimmt.

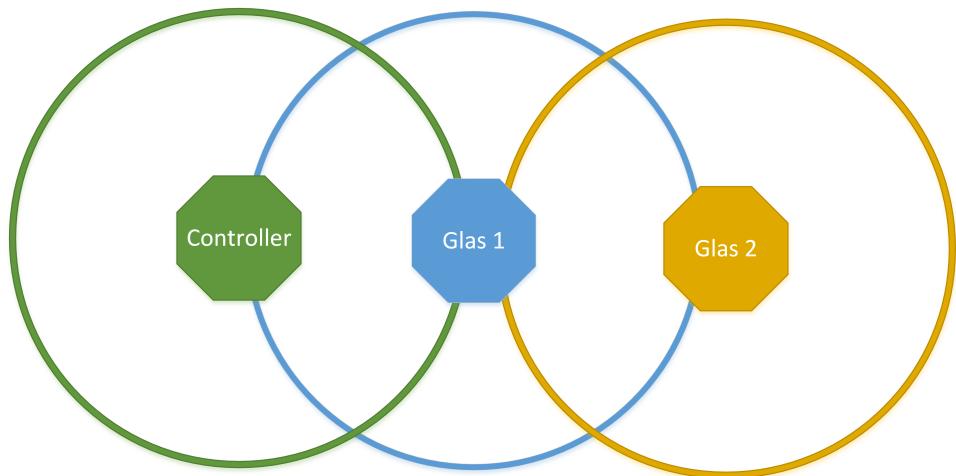


Abbildung 3.20: Beispielhafte Darstellung des sich erweiternden Funk-Netz

In der Abbildung 3.20 ist ein Beispiel für ein solches Netz dargestellt. Die äußeren Kreise symbolisieren hierbei jeweils die Reichweite der einzelnen Komponenten. Offensichtlich kann der Controller nicht das Glas 2 erreichen, da es sich außerhalb seiner Reichweite befindet. Allerdings kann er das Glas 1 erreichen. Dieses wiederum ist in Reichweite zu Glas 2, wodurch der Controller über das Glas 1 Glas 2 erreichen kann.

Um dies zu ermöglichen, wird jedes Paket, welches von einem Glas empfangen wird, direkt weiter geleitet. Damit keine Sendeschleifen entstehen, sind zwei Sicherheitsmechanismen eingebaut:

1. TimeToLive
2. Überprüfung der MessageId

Mit Hilfe der TimeToLive wird eine maximal Anzahl möglicher Sprünge zwischen Sendern und Empfängern festgelegt. Hierdurch wird ein Paket nicht unendlich oft weitergeleitet, da sonst eine sehr hohe Belastung des Netzwerks die Folge wäre. Für Netze mit hoher Dichte an Geräten kann dieser Wert verringert werden. Für räumlich sehr große Netze entsprechend erhöht. Aktuell ist der Wert auf 4 festgelegt.

Bei der Überprüfung der MessageId wird geprüft, ob ein Paket bereits empfangen bzw. weitergeleitet wurde. Sollte dies der Fall sein, wird es nicht weiter bearbeitet. Auch hierdurch wird die Belastung des Netzwerkes verringert und insbesondere Schleifen in der

Kommunikation verhindert. Ein empfangenes Paket wird standardmäßig mit den letzten 8 empfangenen MessageIds abgeglichen.

### 3.4 Gestenanalyse

Um die sozialen Interaktionen des Glas zu ermöglichen, wird unter anderem die Erkennung eines Anstoß zweier Gläser benötigt. Hierfür bietet es sich an, Werte wie die Beschleunigung oder Lage des Glases zu analysieren. Aus diesem Grund ist an den Mikrocontroller das *LSM9DS0*-Modul angeschlossen. Es ermöglicht die schnelle und präzise Erfassung dieser Daten. Des Weiteren gibt es für das Modul bereitgestellte Bibliotheken, welche den Hardwarezugriff abstrahieren.

Für die Bestimmung der Erkennungswerte wurde ein MATLAB-Skript mit einem dazu passenden Arduino-Sketch erstellt. Das MATLAB-Skript erfasst zyklisch, über die serielle Schnittstelle, die aktuellen Werte für das Gyroskop, das Accelerometer und das Magnetometer des *LSM9DS0*-Modul. Diese werden während der Aufzeichnung in einer Kurve grafisch dargestellt. Sobald die relevanten Daten aufgezeichnet sind, kann die Aufzeichnung beendet, die Daten gemittelt, visuell aufgearbeitet und die Gradienten der Funktionen bestimmt werden, um eine möglichst genaue Analyse der Daten zu ermöglichen. Ein Beispiel für eine solche Aufzeichnung der Absolutwerte ist in der Abbildung 3.21 und 3.22 dargestellt. Die X-Achse ist in Abtastpunkte unterteilt.

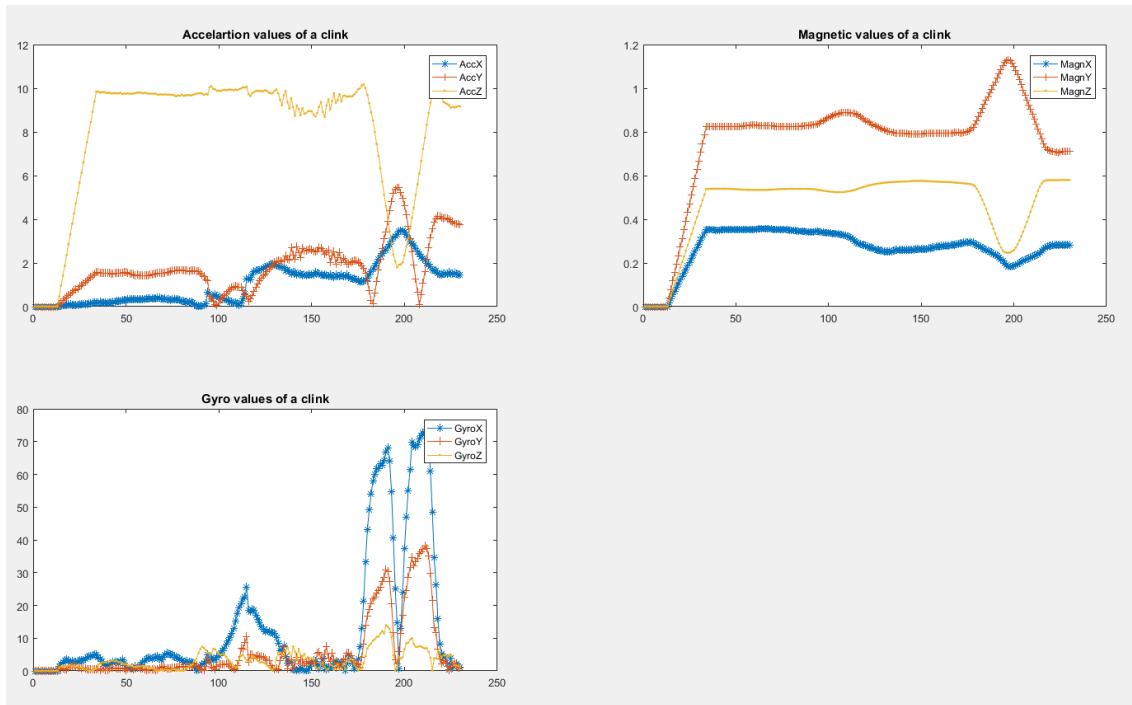


Abbildung 3.21: Aufzeichnung der Absolutwerte des *LSM9DS0*-Modul

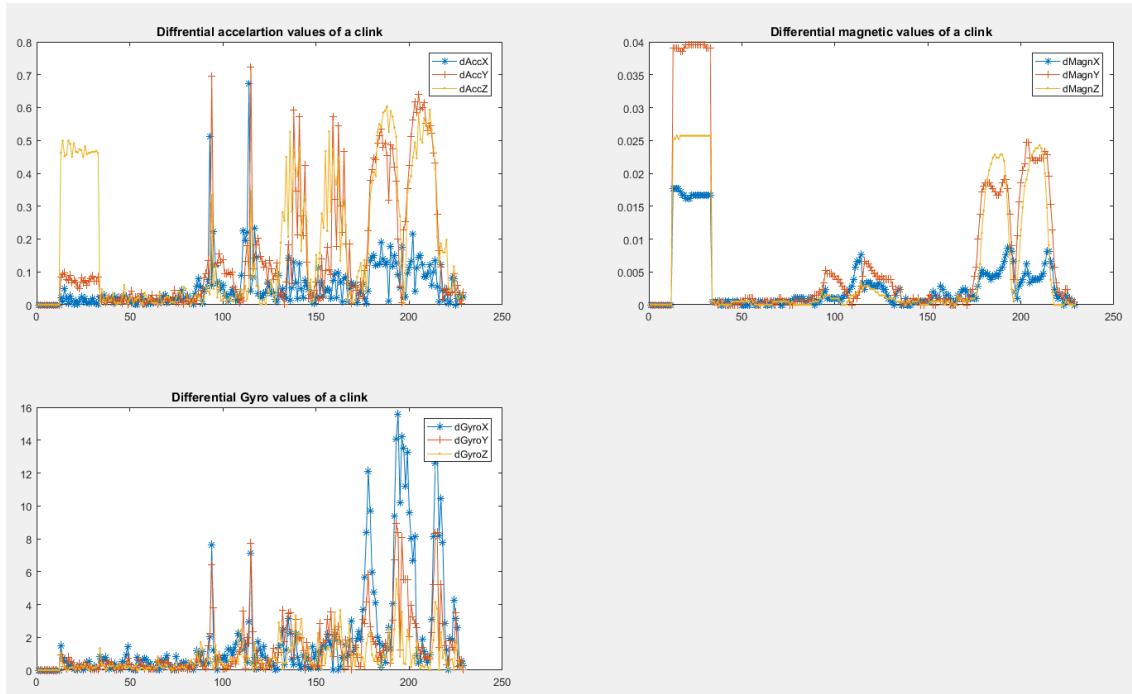


Abbildung 3.22: Berechnete Gradienten der Absolutwerte des LSM9DS0-Modul

In den Abbildungen sind zwei markante Bereiche vorhanden. Zum einen im Bereich von 100, zum anderen im Bereich von 200 Abtastpunkten. Hierbei stellt der Bereich um 200 Punkte den deutlich größeren Ausschlag dar. Der anfängliche Anstieg resultiert aus dem Beginn der Messung.

An dieser Stelle ergibt sich jedoch ein Problem: bei dem stärkeren Ausschlag handelt es sich lediglich um den Vorgang des Trinkens, sprich dem Kippen des Glases. Der Anstoßvorgang um den Bereich von 100 Abtastpunkten herum, ist deutlich weniger signifikant. Ebenso sind die Gradienten der beiden Vorgänge nicht deutlich voneinander zu unterscheiden.

Es lässt sich also erkennen, dass sich das Verhalten der unterschiedlichen Messgeräte, abgesehen von den Wertebereichen, sehr wenig voneinander unterscheidet. Da die Abfrage der Daten und die Menge an zu verarbeitenden Werten proportional zur Anzahl der Messgeräte wächst, wurde entschieden, nur anhand eines Messgerätes die Erkennung der Vorgänge zu realisieren.

Die Beschleunigungswerte des Moduls stellen hierbei die beste Grundlage dar, da sie sich beim Anstoßvorgang in der Z-Achse nur minimal verändern. Lediglich beim Kippen wird in dieser Achse eine starke Änderung hervorgerufen. Somit dient diese als Grundlage für die Erkennung.

Um ein Anstoßen zu erkennen ergeben sich aus den Abbildungen 3.21 und 3.22 folgende Bedingungen:

1. Beschleunigung in Z Richtung größer als 4,14G
2. und einer der Gradienten in X/Y/Z-Richtung größer als 4

Durch diese beiden Bedingungen lässt sich ein Anstoßen erkennen. Das Kippen des Glases zum Trinken soll ebenfalls erkannt werden. Sobald dies erkannt wird, soll das Licht ausgeschaltet werden, um den Benutzer nicht zu blenden. Zur Erkennung dieser Geste ergeben sich die folgenden Bedingungen:

1. Beschleunigung in Z Richtung kleiner als 4,14G

## 2. und Beschleunigung in X oder Y Richtung größer als 8G

Diese Bedingungen entsprechen etwa einem Ankippwinkel von 65 Grad.

Damit das Glas beim Trinken nicht flackert, wurde zum wieder einschalten eine Hysterese eingebaut. So müssen folgende Bedingungen zum Wiedereinschalten erfüllt sein:

1. Beschleunigung in Z Richtung größer als 8G
2. und Beschleunigung in X oder Y Richtung kleiner als 4,14G

Diese Bedingungen entsprechen etwa einem Ankippwinkel von 35 Grad.

## 3.5 Gehäuse und Design

Um den Anforderungen an ein normales Glas zu entsprechen, muss das Gehäuse für die Elektronik am LUMINOGLASS aus zwei Teilen bestehen. Ein Teil ist mit entsprechendem Klebstoff fest am Glas fixiert und hält den anderen, abnehmbaren Teil mit einem geeigneten Mechanismus fest. Die Elektronik ist also nicht fest mit dem Glas verbunden, sondern kann für verschiedene Zwecke abgenommen werden.

Das LUMINOGLASS ist damit (bei niedrigen Temperaturen) Spülmaschinen-geeignet. Für die Mechanik zum Festhalten des Innengehäuses gibt es mehrere Möglichkeiten:

- Dreh-Verschluss
- Gewinde-Verschluss
- Klick-Verschluss

Ein Gewinde könnte sich im Gebrauch von alleine ausdrehen oder sich festfressen. Für einen Klick-Verschluss muss das Material flexibel genug sein, damit nichts herausbricht. Daher ist der Drehverschluss für diese Anwendung am günstigsten. Das Glas wird mit einem Außengehäuse nach unten „erweitert“, welches oben und unten geöffnet ist. Die obere Öffnung lässt das Licht des LED-Rings durch, die Untere dient als Einschub für das Innengehäuse, welches die Elektronik beinhaltet. Ein erster Entwurf könnte wie auf Abbildung 3.23 auf der nächsten Seite realisiert werden.

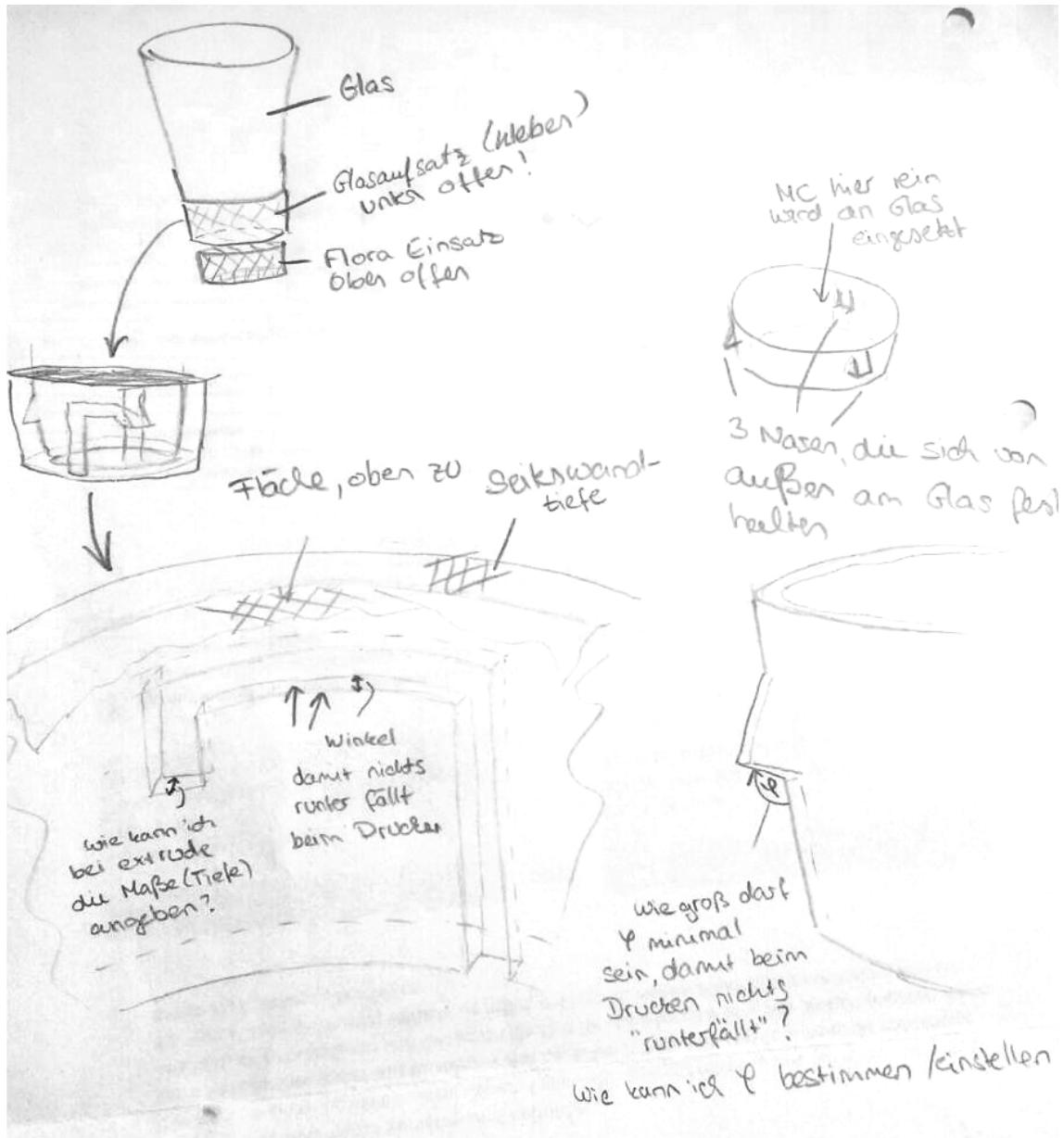


Abbildung 3.23: Gehäuse Skizze

Da die Elektronik rund ist, soll das Innengehäuse einem Zylinder gleichen. Dieser ist oben geöffnet und hat seitlich zwei Nasen zum einhaken ins Außengehäuse. Diese werden dort in zwei Bahnen nach oben, danach zur Seite und anschließend wieder ein Stück nach unten geführt (siehe Abbildung 3.23). Das niedrigere Innengehäuse hängt an den beiden Nasen im höheren Außengehäuse. Das Gewicht des Glases mit Inhalt liegt auf dem Außengehäuse, daher muss eine ausreichende Stabilität gegeben sein. Die Elektronik bestimmt die Maße des Innengehäuses und somit auch die Dicke des Außengehäuses. Dieses soll optisch zu dem verwendeten Glas passen und muss daran angeglichen werden. Als Objekt wurde das Glas auf Abbildung 3.24 auf der nächsten Seite gewählt.



Abbildung 3.24: Verwendetes Cocktail Glas

Die beiden Teile wurden im 3D-Programm *Blender* modelliert und anschließend mit einem 3D-Drucker gefertigt.

**Blender** Das Gehäusedesign wurde mit dem Programm *Blender* (<https://www.blender.org/download/>) an das Glas angepasst. *Blender* ist eine Open Source Software für die Erstellung von 3D-Objekten. Das Programm unterstützt die Möglichkeit zur Vorbereitung eines Objektes für den 3D-Druck und prüft, ob das entsprechende Objekt überhaupt druckbar ist.

**Innengehäuse** Die absoluten Maße der Elektronik dienen als Ausgangsmaße für das Innengehäuse. Plus einer zusätzlichen Toleranz für Stecker und Lötstellen, misst das Gehäuse einen Innenradius von 52,5 Millimeter. Darin findet die Elektronik mit 45 Millimeter Durchmesser ausreichend Platz. Die Elektronik umfasst folgende Bauteile:

- Controller
- LED Ring
- Lagesensor
- Funkmodul
- Akku

Controller und LED-Ring bestimmen mit maximalem Radius den Durchmesser des Innengehäuses. Die Bauteile werden übereinander gelagert und kommen insgesamt auf eine Höhe von ca 25 Millimeter.

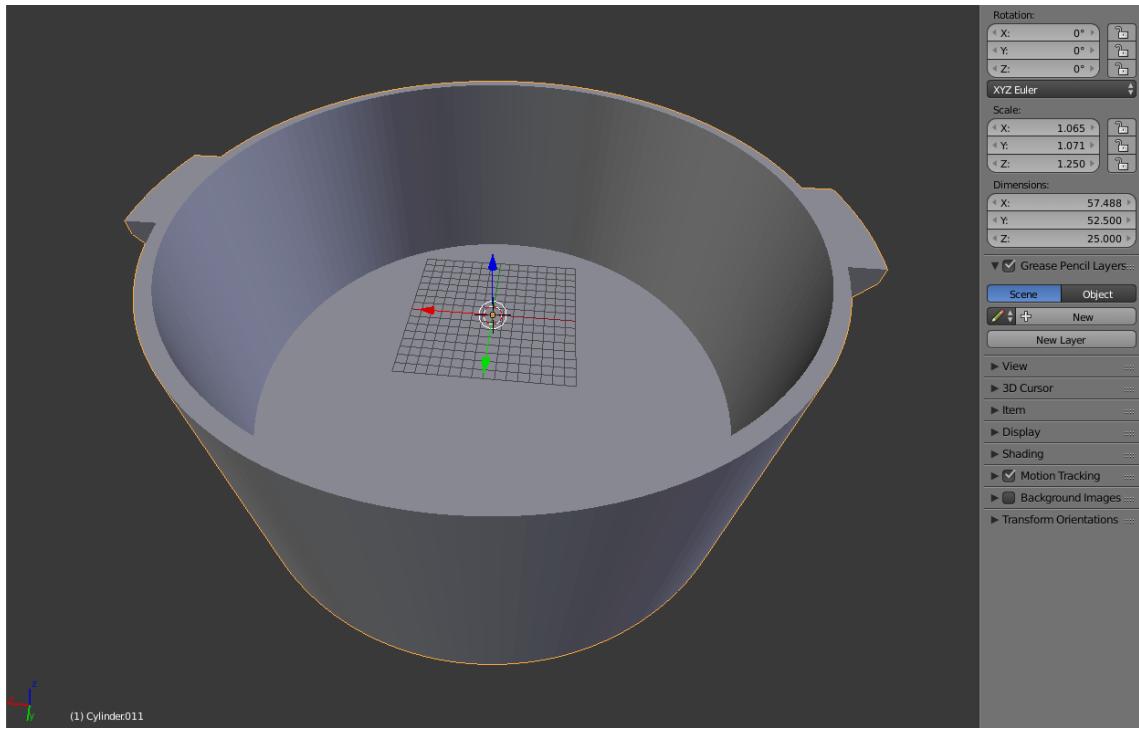


Abbildung 3.25: 3D-Modell Innengehäuse

**Außengehäuse** Das Außengehäuse wird an ein beliebiges Glas angepasst. Das aktuelle Objekt ist ein 9-kantiges Cocktail Glas (siehe Abbildung 3.24 auf der vorherigen Seite). Die Maße am unteren Ende des Glases dienen als Außenmaße. Wichtig ist, dass die Innenseite einen mindestens um 0.5 Millimetergrößerem Radius entspricht, als das Innengehäuse, damit dieses beim Herausschrauben automatisch aus dem Gehäuse „fällt“. Die Maße am Boden des Glases betragen von einer der Kanten 59 Millimeter zur gegenüberliegenden Flachseite. Diese Maße entsprechen denen des Außengehäuses.

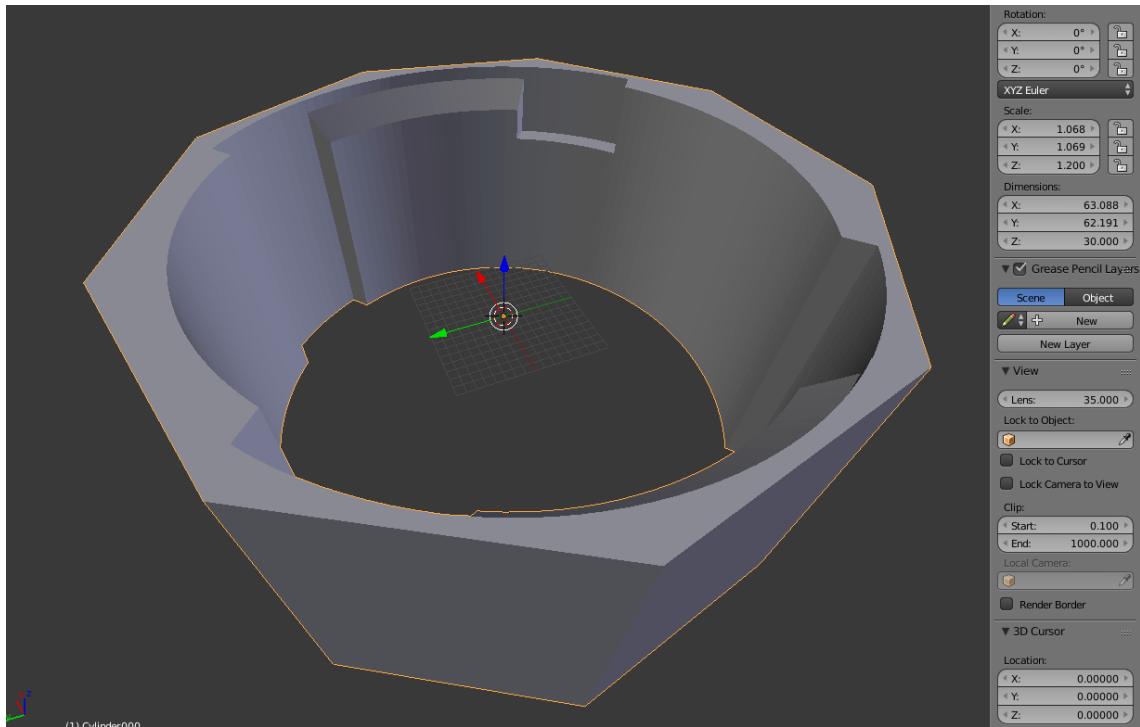


Abbildung 3.26: 3D-Modell Außengehäuse

Die Verschraubung der beiden Teile sieht folgendermaßen aus:

**3D-Druck** Für den 3D-Druck wird das 3D- Objekt als STL-File exportiert. Der 3D- Drucker setzt am Boden eine ca. 3 Millimeterbreite stabilisierenden Struktur um das Objekt herum, damit es während des Druckvorgangs nicht verrutscht. Anschließend druckt er

nacheinander die einzelnen Ebenen des Objektes. Jede Ebene kann nur an Koordinaten gedruckt werden, an denen bereits auf der unteren Ebene eine Schicht Material aufgetragen wurde. Das bedeutet, dass schräge Flächen in einem Winkel von 45 Grad langsam nach außen oder innen aufgebaut werden, da das Material sonst zu Boden fällt. Die Druckdauer variiert stark je nach Aufwand und Größe des Objektes. Die Druckdauer der beiden Teile beträgt je zwischen 20 und 40 Minuten. Um Gewicht zu sparen und die Stabilität zu erhöhen, sind die Teile innen hohl und mit einer Stützstruktur versehen.

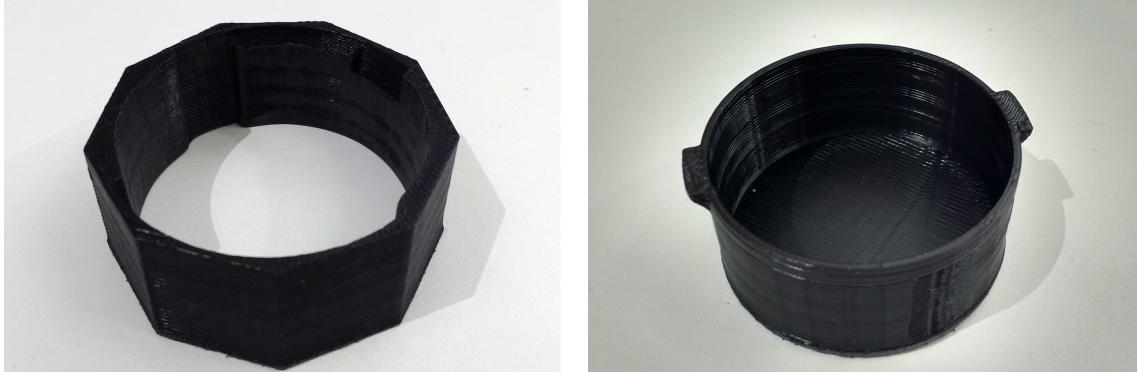


Abbildung 3.27: Ausgedruckte Gehäuseteile

## 3.6 Mikrocontroller Software

### 3.6.1 Glas

#### Initialisierung

1. Watchdog
2. Funkmodul
3. NeoPixel Module
4. Adressen
5. Command-Worker (Abarbeitung der empfangenen Pakete)
6. Liste der bereits empfangenen Paketnummern wird komplett mit 255 gefüllt
7. Starten der Timer zur Lichtsteuerung

**Hauptschleife** In der Hauptschleife wird nach dem Triggern des Watchdogs zunächst überprüft, ob Daten vom Funkmodul vorhanden sind. Sollten Daten vorhanden sein, werden diese ausgelesen und in der Funktion *AnaylzeReceviedPackage* weiter verarbeitet. Am Ende der Hauptschleife wird regelmäßig der Bewegungssensor *LSM9DS0* ausgelesen und die Daten in der Funktion *analyzeGyro* ausgewertet.

```

1 //...
2 if (radio.Available()){ //Check if data available
3     while (radio.Receive((uint8_t*)message.data, &len)){ //read all data
4         flash_center_led(c_success); //flash center led for debug
5         if (len != MESSAGE_SIZE || message.data == NULL){
6             flash_center_led(c_error); //flash center led on error
7         }
8         AnaylzeReceviedPackage(&message); //analyze and use data
9     }
10 } //...
11 if(on)
12     analyzeGyro();
13 //...

```

Quelltext Ausschnitt 3.4: Glas - Hauptschleife

**AnaylzeReceviedPackage** Sobald ein Paket empfangen wird, wird dieses in der Funktion *AnaylzeReceviedPackage* analysiert und weiterverarbeitet. Hierfür werden zunächst die *Source-, Destinationaddress* und *MessageId* des Paketes bestimmt. Anschließend wird in dem Puffer *package\_number\_buffer* geprüft, ob bereits die *MessageId* des Paketes unter den letzten 8 Paketen gewesen ist. Sollte sie bereits vorhanden gewesen sein, wird das Paket ignoriert, um eine unnötige Belastung des Netzwerkes zu vermeiden. Wenn die *MessageId* noch nicht vorhanden war, wird sie in den Puffer mit eingefügt. Hierdurch fällt die älteste *MessageId* automatisch aus dem zyklischen Puffer heraus.

```

1 //...
2 if (destinationAddress > 0){ //Always Forward Packets to Controller
3 //Check if the package was already received
4     if (package_number_buffer.Exists(package->content.Message_Id)){
5         return;
6     }
7     package_number_buffer.Push(package->content.Message_Id);
8 }
9 //...

```

Quelltext Ausschnitt 3.5: Glas - MessageId überprüfen

Wenn es sich bei der *Destinationaddress* um die Broadcastadresse handeln sollte, wird das Paket weiter verarbeitet. Der Aufruf der *Command-Worker* sieht bei jeder Verarbeitung gleich aus, weshalb er an dieser Stelle einmal exemplarisch dargestellt werden soll.

```

1 //...
2 if (package->content.Command_Number < NUM_COMMANDS){ //Check if we can use
    it
3     noInterrupts(); //Disable interrupts
4     cmd_worker[package->content.Command_Number](package); //Call the correct
        cmd_worker
5     interrupts();
6 }
7 //...

```

Quelltext Ausschnitt 3.6: Glas - Command-Worker aufrufen

Sollte die *Destinationaddress* größer als 0 sein, wird zunächst überprüft, ob es sich bei der *Destinationaddress* um die eigene *GUID* handelt. Abhängig von den enthaltenden Payload des *Commands*, wird eine neue *GUID* generiert, oder aber die eigene Adresse *ownAddress* gesetzt.

```

1 //...
2 if (destinationAddress == guid){ //Should be a ping
3     if(package->content.Command_Data_2 > 0){
4         cmd_ping(package);
5         return;
6     }
7     //Guid Valid, adapt address
8     ownAddress = (((long)package->content.Command_Data_0) << 8) | package->
9         content.Command_Data_1;
10    return;
11}
11//...

```

Quelltext Ausschnitt 3.7: Glas - GUID == Destinationaddress

Andernfalls wird mittels des *Address-Workers* überprüft, ob die *ownAddress* anhand des Adressoperators zu der *Destinationaddress* passt. Wenn ja, wird das Paket mit dem passenden *Command-Worker* weiterverarbeitet. Zum Schluss wird überprüft, ob die TimeToLive des Paketes abgelaufen ist. Sollte dies nicht der Fall sein, wird die TimeToLive um eins dekrementiert und mittels der Funktion *SendMessage* weitergeleitet.

**Command Handling** Um schnell auf die *CommandTypen* zu reagieren und gleichzeitig einen übersichtlichen Quelltext zu behalten, ist das *Command Handling* durch ein Feld von Funktionspointern realisiert. In diesem Feld ist für jeden *CommandType* der verarbeitet werden kann, ein Pointer auf eine Funktion vorhanden, welche die spätere Weiterverarbeitung übernimmt. Hierdurch können neue *Commands* durch Erweitern des Feldes einfach hinzugefügt werden.

Die Funktionspointer sind hierbei definiert als:

```
1 typedef void(*command_worker)(DatagrammController2Glass * datagramm);
```

Quelltext Ausschnitt 3.8: Glas - Command Funktionspointer

Innerhalb der Realisierungen der Funktionen zu den *Commands*, wird die Verarbeitung der Pakete durchgeführt. Hier werden z.B. die Farben gesetzt, das Glas eingeschaltet, ausgeschaltet oder andere Effekte realisiert.

**Address Handling** Wie in der Tabelle 3.2 dargestellt, gibt es die Möglichkeit neben der *DesinationAddress* auch einen *AddressOperator* zu versenden. Dieser modifiziert die Handhabung insofern, als das er Operationen wie Prüfung auf Gleichheit, Modulo oder größer/kleiner auf die *DestinationAddress* anwendet. Das Ergebnis dieser Operation wird anschließend in Relation zu der eigenen Adresse gesetzt. Wenn eine Übereinstimmung vorhanden ist, wird das Paket verarbeitet.

Folgende AddressOperator sind derzeit vorhanden:

1. Equal
2. NotEqual

3. Mod
4. NotMod
5. Higher
6. Lower

Eine Erklärung der Operatoren findet sich in Abbildung 3.3.2 auf Seite 25. Die Analyse der Adressen erfolgt ebenso wie das *Command Handling* mit der Hilfe eines Arrays von Funktionspointern. Der aufzurufende Funktionspointer wird entsprechend des *AddressOperator* ausgewählt und ist folgendermaßen definiert:

```
1 typedef bool(*address_worker)(long destination);
```

Quelltext Ausschnitt 3.9: Glas - AddressOperator Funktionspointer

Bei dem *Address Handling* wird auf diese Methode der Verarbeitung aus den gleichen Gründen wie in dem *Command Handling* zurückgegriffen.

**AnalyseGyro** Das Gyroskop zur Gestenerkennung wird zyklisch in der Hauptschleife mit Aufruf der Funktion *AnalyzeGyro* ausgewertet. Hierbei wird auf die von Adafruit bereitgestellten Funktionen der *AdafruitLSM9DS0Library*<sup>8</sup> zurückgegriffen.

Bei jedem Zyklus werden zunächst die Daten per *I2C* aus dem Sensor geladen und anschließend in je einen Puffer für jede Achse eingefügt (X,Y,Z).

```
1 //...
2 sensor.getEvent(&accel, NULL, NULL, NULL);
3 AccXMean[GyroLoopCounter] = accel.acceleration.x; //Same for Y and Z
4 //...
```

Quelltext Ausschnitt 3.10: Glas - Gyro Auslesen

Um die Rechenlast des Mikrocontrollers und die Störanfälligkeit auf einzelne Messwertfehler zu verringern, werden die so gesammelten Daten jeden vierten Zyklus mit einem Mittelwertfilter und einer Betragsfunktion zu geglätteten Werten verarbeitet.

```
1 //...
2 float AccX = abs(CalcMean(AccXMean, ACC_MEAN_LENGTH)); //Same for Y and Z
3 //...
```

Quelltext Ausschnitt 3.11: Glas - Gyro Mittelwertbildung

Mit diesen Werten wird nun die Bewegungsanalyse aus Abschnitt 3.4 auf Seite 29 ausgeführt und das Licht während des Trinkens deaktiviert.

Wenn der Spielemodus aktiviert ist, wird zusätzlich eine numerische Differenzierung vorgenommen, um die Gradienten der Signale zu bestimmen.

```
1 //...
2 dAccX = abs(lastdAccX - AccX); //Same for Y and Z
3 lastdAccX = AccX;
4 //...
```

Quelltext Ausschnitt 3.12: Glas - Bewegungsanalyse, numerische Differenzierung

Unter Zuhilfenahme dieser zusätzlichen Daten, kann anschließend ein Anstoßvorgang ermittelt werden. Damit hierbei keine Flut von möglichen Anstoßvorgängen entsteht, ist eine zeitliche Sperrre eingebaut. Es kann nur alle 500ms ein Anstoßen erkannt werden.

---

<sup>8</sup>adafruit/Adafruit\_LSM9DS0\_Library.

**Color Handling** Die Ansteuerung der *NeoPixel-Module* auf dem *NeoPixel-Ring* wird durch die Verwendung von Funktionen aus der *Adafruit\_NeoPixel-Library*<sup>9</sup> realisiert. Damit Funktionalitäten wie das periodische Blinken der LEDs sowohl einfach, als auch präzise ausgeführt werden können, wird die Funktion *setLed*, welche die Ansteuerung der LEDs übernimmt, zyklisch in dem Interrupt des Timers 2 ausgeführt. Der Timer 2 wird mit einer Frequenz von 50Hz betrieben, was saubere und nicht-abgehackt wirkende Übergänge zwischen Farben ermöglicht.

Welche Einstellungen vorgenommen werden sollen, wird hierbei durch die *CommandWorker-Funktionen* gesetzt. Dies geschieht durch setzen globaler Zustands-Variablen.

```

1 //...
2 for (unsigned char led = 0; led < NUM_LEDS; led++){
3     if (!strobeActive || strobeToggle){ //Not Strobe
4         color = currentPixelColors[led]->toUInt32(&current_brightness);
5         position = led;
6         if (rotationActive){ //rotation
7             if (rotationDirection == AgainstClock){
8                 if (position - rotationOffset < 0){
9                     position = NUM_LEDS + (position - rotationOffset);
10                }
11            else{
12                position -= rotationOffset;
13            }
14        }
15    else{
16        position += rotationOffset;
17        position %= NUM_LEDS;
18    }
19 }
20 strip.setPixelColor(position, color);
21 }
22 else //Strobe Black
23 strip.setPixelColor(led, 0);
24 }
25 strip.show();
26 //...

```

Quelltext Ausschnitt 3.13: Glas - Ansteuerung der NeoPixel-Module

Die Farben der globalen Zustands-Variablen werden in der *setLed-Funktion* angewendet. Neben den Farben werden hier auch Berechnungen zur Rotation und Farbverläufen realisiert. Sobald diese abgeschlossen sind und bestimmt wurde, welche *NeoPixel-Module* wie leuchten sollen, werden die Informationen in der Funktion *drawPixels* an den *NeoPixel-Ring* gesendet.

```

1 //...
2 cSet.set(r, g, b, w);
3 //...

```

Quelltext Ausschnitt 3.14: Glas - Beispiel zum Setzen von Farben für die NeoPixel-Module

---

<sup>9</sup> adafruit/Adafruit\_NeoPixel.

### 3.6.2 Transmitter

Im Rahmen der Hauptschleife des Transmitters wird zunächst eine serielle Verbindung aufgebaut. Anschließend wird das Funkmodul initialisiert und der Watchdog eingeschaltet. Dieser setzt den Mikrocontroller bei einem potenziellen Fehler automatisch zurück. In der Hauptschleife wird zunächst überprüft, ob ein Datenpaket über das Funkmodul vorhanden ist. Wenn ja, wird dieses über die serielle Schnittstelle an den Controller gesendet.

```
1 //...
2 if (radio.Available()){
3     while (radio.Receive(message, &len)){
4         if (message != NULL && Serial){
5             Serial.write(message, len);
6             Serial.flush();
7         }
8     }
9 }
10 //...
```

Quelltext Ausschnitt 3.15: Transmitter - Überprüfen ob Daten über Funk vorhanden sind

Sollten keine Daten vorhanden sein, wird überprüft, ob der Controller Daten über die serielle Schnittstelle zur Verfügung stellt.

```
1 //...
2 if (Serial.available() >= MESSAGE_SIZE){
3     uint8_t bufferWriteIndex = 0;
4     while (Serial.available() && bufferWriteIndex < MESSAGE_SIZE){
5         inputBuffer[bufferWriteIndex] = Serial.read();
6         bufferWriteIndex++;
7     }
8     SendData(inputBuffer);
9 }
10 radio.SetModeRx();
11 //...
```

Quelltext Ausschnitt 3.16: Transmitter - Überprüfen ob Daten über Seriell vorhanden sind

Sollten Daten vorhanden sein, werden diese gelesen und mittels *SendData* über das Funkmodul versendet. Anschließend ist der Transmitter wieder Empfangsbereit.

```
1 //...
2 while(!radio.CanSend()){
3     YIELD    //Wait till we can send
4 }
5 radio.Send(message, MESSAGE_SIZE * sizeof(uint8_t)); //Send Data
6 if (!radio.WaitSendFinished()){//Wait until sending is finished.
7 }
8 //...
```

Quelltext Ausschnitt 3.17: Transmitter - Daten über Funk senden

## 4 | Ausblick

Zur Verbesserung der Optik des LUMINOGLASS kann die Elektronik verkleinert werden. Hierzu ist es von Nöten, eine eigene Platine ätzen, maschinell bestücken und löten zu lassen. Somit würde diese die minimale Höhe erreichen, sodass das Gehäuse nur noch wenige Millimeterhoch ist. Leider belaufen sich die Kosten für die Einzelanfertigung einer solchen Platine auf einen Preis von ca. 300€ pro Stück. Daher wird diese Optimierung in diesem Projekt nicht möglich sein. Weiterhin könnte die Elektronik direkt in ein spezielles Glas eingebaut werden und ein Aufladen des Akkus über Induktion erfolgen. Außerdem könnte ein spezieller, runder Akku verwendet werden um die Laufzeit des Glases zu verlängern.

Statt jedes Glas gleichzeitig als Sender und Empfänger zu betreiben, wäre es in Systemen mit großer Teilnehmerzahl sinnvoll eigene Repeaterstationen in definierten Abständen zu verteilen, um Latenzzeiten zu vermindern und die Übertragungssicherheit zu steigern.

In der Controllersoftware könnten weitere Spiele und im Glas weitere Gesten implementiert werden, insofern ein Mikrocontroller mit einem größeren Flash-Speicher verwendet wird.

# Literatur

- [1] *adafruit/Adafruit\_LSM9DS0\_Library*. URL: [https://github.com/adafruit/Adafruit\\_LSM9DS0\\_Library](https://github.com/adafruit/Adafruit_LSM9DS0_Library) (besucht am 05.02.2017).
- [2] *adafruit/Adafruit\_NeoPixel*. URL: [https://github.com/adafruit/Adafruit\\_NeoPixel](https://github.com/adafruit/Adafruit_NeoPixel) (besucht am 05.02.2017).
- [3] *FLORA - Wearable electronic platform: Arduino-compatible [v3]* ID: 659 - \$14.95 : Adafruit Industries, Unique & fun DIY electronics and kits. Englisch. Online-Shop. URL: <https://www.adafruit.com/products/659> (besucht am 04.02.2017).
- [4] *FLORA 9-DOF Accelerometer/Gyroscope/Magnetometer - LSM9DS0 [v1.0]* ID: 2020 - \$19.95 : Adafruit Industries, Unique & fun DIY electronics and kits. Englisch. Online-Shop. URL: <https://www.adafruit.com/product/2020> (besucht am 04.02.2017).
- [5] *NeoPixel Ring - 16 x 5050 RGB LED with Integrated Drivers* ID: 1463 - \$9.95 : Adafruit Industries, Unique & fun DIY electronics and kits. Englisch. Online-Shop. URL: <https://www.adafruit.com/products/1463> (besucht am 04.02.2017).
- [6] *NRF24 Mini SMD Wireless Module (NRF24L01 Compatible)*. Englisch. Online-Shop. URL: <http://www.electrodragon.com/product/nrf24-mini-smd-wireless-module-nrf24l01-compatible/> (besucht am 04.02.2017).

## **Verwendete Fremdkomponenten**

1. NAudio Ms-PL License <https://naudio.codeplex.com/>
2. MahApps Metro - MIT License <http://mahapps.com/>
3. OxyPlot - MIT License <http://www.oxyplot.org/>
4. ColorWheel Control and Tools - BSD License, Andrew Syrov <http://color.codeplex.com/>
5. Icons - Open Font License 1.1 <https://materialdesignicons.com/>
6. Adafruit NeoPixel Library - LGPL-3.0 License [https://github.com/adafruit/Adafruit\\_NeoPixel](https://github.com/adafruit/Adafruit_NeoPixel)
7. Radiohead Library - GPL Version 2 License, Mike McCauley <http://www.airspayce.com/mikem/arduino/RadioHead/>
8. Adafruit LSM9DS0 Library - BSD License, Kevin Townsend [https://github.com/adafruit/Adafruit\\_LSM9DS0\\_Library](https://github.com/adafruit/Adafruit_LSM9DS0_Library)
9. Light Bulb Icon - Flaticon Free License [http://www.flaticon.com/free-icon/light-bulb\\_178395](http://www.flaticon.com/free-icon/light-bulb_178395)